コンピュータ・サイエンス研究書シリーズ ④

オペレーティング・システムへの構造的アプローチ（中巻）

日本アイ・ビー・エム㈱
江村潤朗 著

日本コンピュータ協会

コンピュータ・サイエンス研究書シリーズ

オペレーティング・システムへの構造的アプローチ（中巻）

日本アイ・ビー・エム㈱　江村潤朗 著

日本コンピュータ協会

まえがき

オペレーティング・システムが一般化したのは，第3世代のコンピュータからです．確かに，それ以前のコンピュータでも大型のものであるとか，技術計算用のコンピュータではオペレーティング・システムに近いソフトウェア・システムが用いられていました．しかし，その主眼はジョブの連続処理と入出力オペレーションの効率的制御にあり，今ほど体系化されたソフトウェア群ではありませんでした．ところが，第3世代以後のコンピュータではシステムの全体的な生産性を向上させることを目標に，制御プログラムと処理プログラムとで構成されるオペレーティング・システムが全く普通のものとなってしまいました．

情報処理システムの生産性を向上させるためには，それにかかわりを持っているシステム資源を有効に活用することです．言うまでもなく，システム資源としては，CPUタイム，主記憶装置の容量，入出力チャネル時間と入出力装置などで代表されるハードウェア資源，CPUで処理するデータとCPUに対して特定順序で処理を行なうように指示する一連の命令からなるプログラムとで代表される情報資源，それにシステムを構築し使用する人的資源があります．情報処理システムの生産性は，これらハードウェア資源，情報資源および人的資源を上手に組み合わせ，それぞれの仕事を行なうのにどのように巧妙な活用をするかにかかっているのです．このために，オペレーティング・システムが出現したのです．オペレーティング・システムは，情報処理システムの性能を高め，システムの使い易さを増大させることを根源的目標にしています．システムの性能は，スループットと応答時間で評価されます．また，使い易さはシステムをいかに容易に使用できるかを表わす尺度です．使い易さは，ユーザがプログラミング上の補助手段，種々のサービスや前もってコーディングされたルーチンなどを簡単に活用できるようにすることで達成されます．ともあれ，スループット，応答時間，使い易さといった諸要素のからみによって，システ

ム全体の生産性を向上させることができるのです．

このような目標を果たすための制御プログラムの機能として，データ管理機能，ジョブ管理機能およびタスク管理機能があります．これが，オペレーティング・システムの三大基本機能です．第3世代のコンピュータが出現してから十年余りが経過しましたが，オペレーティング・システムの基本機能はそのまま生きつづけており，強化されてきています．種々の新たな機能が付加され，拡張されてきたとはいうものの，データ管理，ジョブ管理，タスク管理の機能が主軸になっていることに変りありません．今後もこれら3つの機能がオペレーティング・システムの主要な役割りを果たしてゆくことでしょう．

本書では，上巻の第5章でデータ管理の機能を取り上げて解説した内容をふまえたうえで，第6章「**データ・セットの編成法とアクセス方式**」で，データ管理機能のハイライトであるデータ・セットの編成法とその処理の仕方について言及しました．順次データ・セット，区分データ・セット，索引順次データ・セットおよび直接データ・セットの4つの編成法を取り上げ，それぞれのアクセスの仕方を具体例にもとづいて解説しています．

第7章「**ジョブ管理の諸機能とその構造**」では，ジョブ管理の機能について，歴史的発展過程を考察したうえで，ジョブ管理の具体的機能構造を明らかにしています．順次式ジョブ・スケジューリング・システムと優先式ジョブ・スケジューリング・システムの構造を解明したあとで，それらの機能をより具体化する目的でジョブ制御ステートメントでの指定内容についても言及しています．

なお，**下巻**では「**タスク管理の諸機能とその構造**」および「**仮想記憶システムの概念と構造**」に関して詳述しています．

最後に，業務多忙と病気入院のため本書の執筆が大幅に遅れてしまい，読者の皆様および**日本コンピュータ協会**の村越勝弘編集長をはじめとしたスタッフの方々にご迷惑をおかけしましたことをおわびいたします．また，この間終始協力し援助してくれた，今は亡き愛妻芳子の心労と配慮に感謝すると同時に，心をこめて本書を霊前に捧げたい．

昭和50年6月

著　者

目　　次

第 6 章

データ・セットの編成法とアクセス方式

一. はじめに

今から十数年前，コンピュータが産業界に初めて導入された．その当時は，多数の人間が手作業で処理していた定型的な仕事を解決するのにコンピュータが活用される場合が多かった．給与計算，経理事務，統計事務や各種受払い業務などの手作業事務の事務処理を代行するのがその主要な役割りをなしていた．その後，生産管理，在庫管理，原価管理，原価計算，在庫調整などのための資料を作成するといった生産や原価などの統合化による事務の事後処理のためにコンピュータが利用されるようになっていった．これらの業務の自動処理は，コンピュータの正確性と処理時間の短縮といった面での優秀性を十分に証明した．

その結果，コンピュータ利用はもっと高度で複雑な新しい分野の拡大へ挑戦するようになった．手作業の単なる機械化であるとか，事務の事後処理よりもはるかに程度が高いシステム的な適用業務を開発しはじめた．たとえば，経営情報システム，プロセス制御システム，流通制御システム，医療診断システム，学習制御システム（CAI），情報検査システムなどがそれである．

このようにコンピュータが急速に発展した結果，コンピュータ部門で処理すべき適用業務の数が非常に増加したばかりでなく，大規模な適用業務の範囲が拡がりかつ複雑さも一段と増した．これらの問題に対処するために，コンピューティング・システムにかかわりを持つ各種資源，つまり，ハードウェア資源，

ソフトウェア資源および人的資源の効果的活用が追求されることとなったのである.

コンピュータ室の諸資源は多大な投資に相当する. したがって, これら資源を最大限有効利用することによって, 数多くのかつ複雑・高度な適用業務を消化することに対処するような方向を目指ざすこととなる. そのためには, 各資源をそれぞれ最も適した仕事にたずさわらせ, それらが本来もっている能力または性能を 100 パーセント近く発揮できるような工夫が要求される. CPU にデータの処理という極めて単調な作業を休むことなく行わせるためには, ハードウェア資源やソフトウェア資源（単にプログラムだけでなくデータも含めて）が常時利用できるようにしておかなければならない. そのためにジョブの連続処理とか多重プログラミングの構想がでてきた. ハードウェア的な配慮としては, 命令の併行処理だとか高速バッファの考えが組み込まれてきた.

システム設計者, プログラマーあるいはオペレーターなどといった人的資源をもっと創造的な仕事, たとえば新しい適用業務を考えたり, 問題解決のシステムを作ったり, 変化や予期しない事態に対処したり, コンピュータ室をもっと有効に管理したりすることに従事させるには, コンピュータで代行できる仕事から可能なかぎり解放してゆくことが要求される.

コンピュータ部門の生産性は, ハードウェア, ソフトウェアおよび人的資源が効果的に選択・配置され, かつ本来の能力を発揮してそれぞれの仕事が行えるような状況におくことにかかっている. 現代のコンピュータは, わずか数分間で何十億回ものオペレーションを実行できる. しかも, 人間よりもはるかに高い信頼度で行うことができる. しかし, コンピュータが遂行するすべての仕事を企画し, その仕事の実行中に生じるあらゆる偶発事態を処理する責任は, 人間にある. 要するにコンピュータは, 人間の作り出した命令に忠実に従うだけである. プログラムがなければ, コンピュータ・システムといえども何の役にも立たないことは今更いうまでもないことである.

しかし, プログラムがあればコンピュータに思いもよらないことをやらせることもできる. 単に入力データを処理して出力データを作り出す以上のもっと多くの高度なことを行うことができる. コンピュータの数少ない命令をいろい

ろ組み合わせたり，順序を変えたりして，かって人間の力ではできなかったような様々な事柄をコンピュータに行わせるプログラムが作れる．コンピュータにかかわりを持つ諸資源の管理，ジョブのスケジューリングと監視，データやプログラムの検査，入出力装置の操作や制御あるいはプログラム言語の翻訳などといったことを行うプログラムが作れる．

この種の操作をコンピュータに指示して，システム資源の有効利用を計るためのプログラム群がオペレーティング・システムであった．オペレーティング・システムは，コンピュータの高度なシステム適用業務の１つであり，プログラムとデータとが組織的に統合化された集合体である．他のシステム適用業務と同じようにオペレーティング・システムも複雑な適用業務を取り扱うために作られたものである．しかし，通常のシステム適用業務は，コンピュータ室以外の専門的な業務，たとえば銀行業務，プロセス制御，ミサイル設計などといった本来のビジネス業務である．一方，オペレーティング・システムはコンピュータ室自体の業務を処理するために作られたものである．つまり，オペレーティング・システムは，他の適用業務の性能を向上させ，それの実行を制御することを目的に特別に設計されたコンピュータのシステム適用業務といえる．オペレーティング・システムの基本的目標は，システム資源の有効利用を計り，コンピュータの使い易さを援助することによって，データ処理システム全体の生産性を増大させることにある．生産性は，ハードウェアとソフトウェア資源の性能と便利性によって決まる．そして，これらの要素にコンピュータ室の要員の能力が結びついたとき，コンピュータ室全体の生産性が決定するといえる．

ところで，オペレーティング・システムは，それを構成する制御プログラムのもとに，すべての処理プログラムを支配することによってその目標を達成する．制御プログラムと処理プログラムとでオペレーティング・システムが構成されるわけだ．単にこれらプログラムだけでなく，各種の制御テーブルはじめとしたデータもその支配下におく．処理プログラムには，言語翻訳プログラム，連係編集プログラムやユーティリィティ・プログラムなどのサービス・プログラムおよびユーザーが作成した各種の適用業務プログラムがふくまれる．換言

するならば，すべての生産的なプログラムは制御プログラムのもとで機能するということになる．制御プログラムが，すべての処理プログラムを自己の管理のもとにおくことによって，システム資源の有効利用とシステムの使いやすさを実現しようとしているのである．

さて，この制御プログラムであるが，その機能は3つに大別できた．データ管理機能，ジョブ管理機能，タスク管理機能である．

データ管理機能は，データやデータ・セットを標準的な手法で統一的な管理を行う機能であった．データ・セットの編成や処理を標準的な手法で行うだけでなく，データ・セットのカタログを作成しその管理を行うことにより，データを主体としたソフトウェア資源の有効利用と使い易さを計ろうと企図している．また，外部記憶装置のスペースの割り当てと管理およびチャネル・プログラムのスケジューリングと入出力割込みの処理を効率的に達成することによってハードウェア資源の効果的利用をうながす．

ジョブ管理機能は，ジョブを連続的に処理することを主要な機能としている．ジョブとジョブの間に生じるハードウェアやソフトウェアの遊休時間を排除することによって，システム資源の無駄な遊びをなくする．これと同時にオペレーターが行う機械操作の代行や，管理者の行うジョブ・スケジューリングやジョブ管理を代行することによって人的資源にも貢献する．

一方，**タスク管理の機能**は，割込み原因の分析と処理を行うことによってタスクの実行過程を制御し，多重プログラミングを実現する．また，主記憶装置の効果的割り当ても行う．この機能によって，CPU と主記憶装置を主体としたハードウェア資源の無駄を極力なくしてゆくように機能している．

これら制御プログラム機能のうち，データ管理の機能については上巻の第5章「データ管理の諸機能とその構造」で詳述してきた．第5章で取扱った内容を要約したうえで，第5章で論述し得なかったデータ管理の機能について説明してゆくことにしよう．

コンピュータ利用が急速に発展し，コンピュータの適用業務が拡大してゆくと，必然的に取扱うデータも急増する．その結果，企業システムを動かすための基本的情報の大部分がコンピュータ室に集中してくる．かっては，紙テープ

や穿孔カードに記録されていた大量のデータが磁気ディスクや磁気テープなどに記憶されるようになった．これによって，データを記録・保管するためのスペースは減ったが，データ量が爆発的速度で増加し集中化されたがゆえに，それを効果的に管理し使用してゆくうえでの諸々の問題が生じた．

大量のデータを管理するための効果的な体系ができていないことには，データ間で非常に多くの重複が出てしまう．ほとんど同じデータ・セットがいくつも別々に作成され，保管・維持されることも珍らしくない．これは，単に無駄なデータを増加させるだけでなく，データを絶えず最新のものにし，その使用を効果的にコントロールする上にもむずかしい問題をなげかけることになる．また，新しいデータが増えるにつれて，廃物と化した古いデータが沢山残ってゆくことにもなる．これでは，これらデータの適切処理を委託されているコンピュータ部門の人々の身動きがとれなくなってしまいかねない．

さらに，磁気テープや磁気ディスクのようにデータを磁化によって記録する方法が一般化したことによる問題も生じた．それらに記録されているデータは人間の目で見ることが出来ないという理由から，データを分類したり，カタログしたり，探し出したりあるいはデータを利用したり廃棄したりのコントロールのためや，データを記録しておくスペースを探して割り当てたりするためにコンピュータ室の要員が何らかの方法を考え出して，それらの状況をいちいち記録しておかなければならない．この種の作業には，手作業による誤りや，要員の移動に伴って生じる問題がある．

このようにコンピュータ室で取扱うデータの急増と集中化に起因する諸問題以外の問題もある．技術の進歩とコンピュータ利用の発展とが促がした結果として，入出力装置が多様化してきたことだ．これら入出力装置のそれぞれのハードウェア特性を考慮しつつプログラミングを行うことは，プログラマーにとってかなりの負担となる．それに新しい入出力装置が出現するつどそれに合わせてプログラムの書き換えを行うことは，プログラムの本数が増えているだけに大変なロードとなりかねない．そのうえ，入出力装置の本来の性能を生かすようなプログラムを書きあげることも容易ではない．

コンピュータ室におけるデータの増加と集中化および入出力装置の多様化に

よって生じる諸問題を解決し，かつ，これによって与えられる機会を効果的に利用してゆくには，標準的な手続きで統一的に管理してゆく必要がある．この要求を満たすためのものが，オペレーティング・システムのデータ管理機能であった．その機能を図式化してまとめたのが**図6-1**である．

コンピュータ室に集中化された沢山のデータ・セットを管理するものとして，**データ・セットの制御機能**があった．各種のデータ・セットの登録目標であるカタログと各々のデータ・セットの所在をつきとめるためのラベルとを関係づけることにより，データ・セットにつけられた名前でもってデータ・セットを識別したり，追加したり，あるいは削除したりする機能である．また，同一の属性をもったデータ・セットで一定周期で作成されてゆくようなデータ・セットに対しては，同一のデータ・セット名で管理するための世代別データ群という構想を導入した．集中化されたデータ・セットの中には，企業の機密情報や個人のプライバシーにかかわりをもつ情報も少なくない．これら情報は，誰れでもが自由に取扱える性質のものではない．資格ある限られた人々にしかその取出しや変更を認めることはできない．そのための機能として，パスワードによるデータ・セットの保護という手法が組み込まれている．

多様化した入出力装置の実際の入出力オペレーションの制御と入出力オペレーションの完了やエラー処理の面倒をみる入出力監視プログラムの機能もある．この機能は，第3章の八節「入出力オペレーションと監視プログラムの働き」で説明したように，入出力オペレーションを標準的な手法で監視することによって，入出力オペレーションの処理効率をあげると同時にプログラミング上の負担を最少限にくいとめることにあった．

外部記憶装置である磁気ディスクや磁気ドラムのような直接アクセス・ボリュームは，今では欠くことのできないデータ・セットの記憶媒体である．しかし，これら媒体の記憶スペースを手作業で管理することはきわめて厄介であるだけでなく時間を要する．そのうえ，効率的な管理はおぼつかない．この面倒な仕事を一手に引受けてくれるのが，DASDスペース管理ルーチンの機能であった．データ・セットの新規作成や追加によるスペースの要求と不用になったデータ・セットのスペースの明け渡しということが錯綜する中で，DASD上の

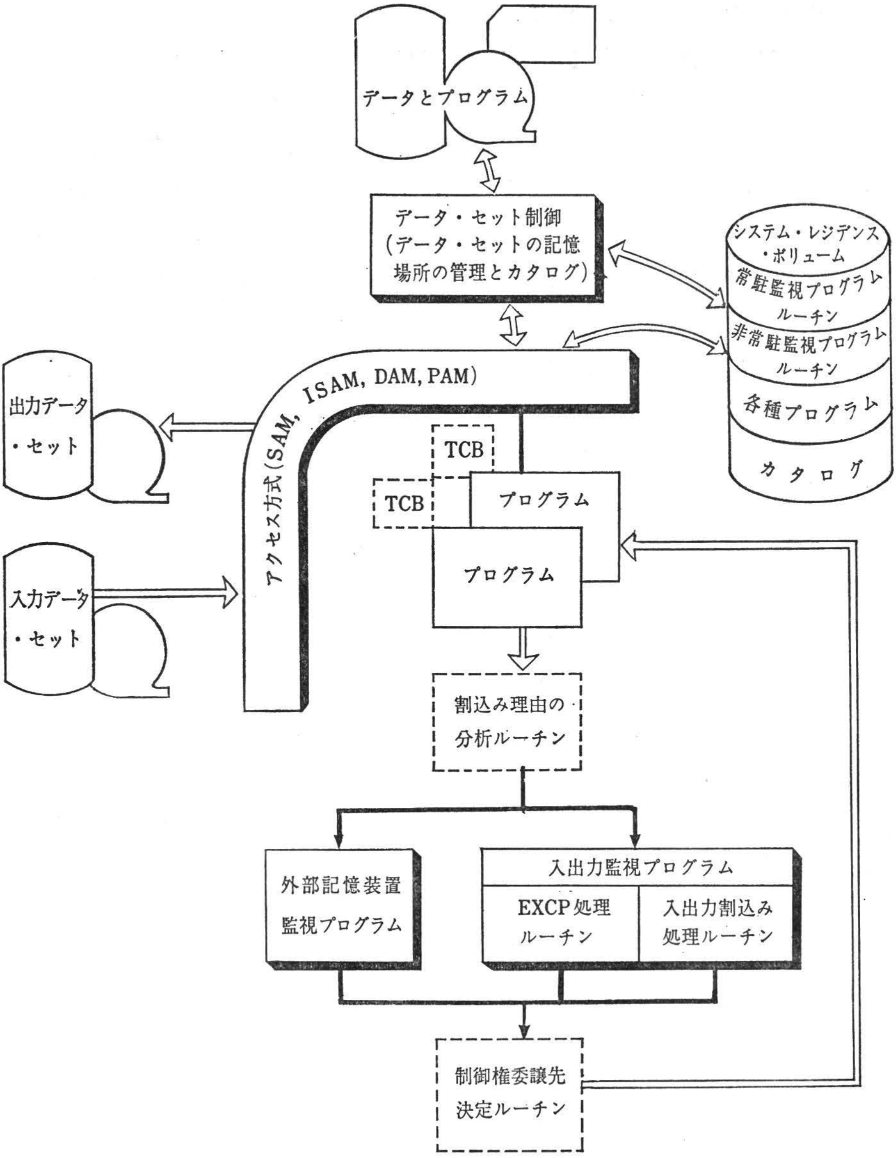


図 6-1　データ管理の機能構造

VTOCと呼ばれる特別なデータ・セットを使うことによって効率的な管理を行ってくれる監視プログラムである．

データ管理の機能のうちで見落すことのできないものとして，データ・セットの編成やデータの記憶と取出しを標準的な手法で行うためのデータ・アクセス機能がある．データ・セットの編成や処理にあたっての中枢的な情報は，データ制御ブロック（DCB）に集約される．このDCBは，処理プログラムの一部を構成するものであるが，ここに貯えられる情報はプログラム作成時に指定できるだけでなく，プログラムの実行時にジョブ制御ステートメントの1つである DD ステートメントやデータ・セット・ラベルからも与えることができる．プログラムの実行時に，データ・セットの属性値がDCBに与えられることは，それだけ融通性に富んだデータ・セットの処理が実現できるのである．特定の入出力装置を念頭におかないで原始プログラムが作成できるのも，この結果である．プログラムの入出力装置からの独立性が実現可能なのは，DCBへの情報が実行時に与えられるからに他ならない．

また，入出力バッファの確保や割り当ての機能もデータ・アクセス機能の1つである．特別なマクロ命令でバッファを確保することもできるし，OPEN命令で自動的に確保することもできる．その時々の条件に合致した手法を選ぶことができるのである．また，バッファの割当てに関してもいくつかの手法が用意されている．

入出力のアクセス方式ルーチンの決定にあたっては，OPEN ルーチンが機能する．アクセス方式ルーチンのどれを選択するかは，DCB の中に示されているデータの編成法，バッファ割当て手法，アクセス手法および入出力装置の諸特性にもとづいて決定する．OPENルーチンは単にアクセス方式ルーチンを選択してローディングするだけでなく，データ・セットの入出力オペレーションを可能な状態にするための様々な前準備を行ってくれる．一方，CLOSE ルーチンは，データ・セットの入出力オペレーションが終了したあとでの最終処理の機能を果すし，EOV ルーチンは入出力オペレーションの過程でボリュームの終りやデータ・セットの終りの状態になったときの処理機能を果してくれる．これら OPEN，CLOSE および EOV ルーチンの機能は，データ・セット

のアクセスに欠くことのできない補助的機能である．これらを総称して，**入出力サポート・ルーチンの機能**と呼ぶ．

以上のデータ管理の機能に関して詳しく説明したのが，第5章「データ管理の諸機能とその構造」であった．ところが，データ管理の機能のうちのデータ・アクセスに関しては，その編成法に関してもアクセス方式に関しても具体的な言及は行っていない．そこで，本章では「データ・セットの編成法とアクセス方式」について論じてゆく．

データの記憶と取出しを決定する要素には2つある．その1つは，データ・セットの編成法であり，他の1つはデータのアクセス技法（Queued か BASIC か）である．この両者の組合わせによって，入出力データ・セットの**アクセス方式**（access method）が決定される．そのうえで，**表 6-1** に示したように該当するアクセス方式ルーチンを使って入出力データ・セットの作成，更新，データの取り出しが行われるのである．これらアクセス方式ルーチンは，処理プログラムに組み込まれるのではなく独立したルーチンとして制御プログラムの一部を構成し，処理プログラムにデータの取り出しや記憶のサービスを提供するのである．

ところで，データの取り出しや記憶にさいして制御プログラムがかかわりをもつデータの最小単位は通常レコードである．そこで，本章ではまず制御プログラムで取扱うことのできるデータ・セットのレコード形式について明らかにする．そのあとで，データ・セットの編成法とアクセス方式について言及してゆくことにする．

データ・セットの編成方法	アクセス方式	
	Queued（待機）	Basic（基本）
順次編成（Sequential）	QSAM	BSAM
索引順次編成（Indexed Sequential）	QISAM	BISAM
直接アクセス編成（Direct Access）		BDAM
区分編成（Partitioned）		BPAM

表 6-1　データ・セットの編成方法とアクセス方式

二. データ・セットのレコード形式

データ・セットは，一般に相互に何らかの論理的関係をもったレコードの集合体である．そして，レコードは処理プログラムが用いる情報の基本単位である．特別な場合には1桁の文字から構成されることもあり得るし，ある商取引の全ての情報から構成されることもあれば，ある実験の任意の時点のパラメータで構成される場合もある．いずれにしろ，たいていのデータ処理では，個々のレコードを読取り，それを処理し，結果を出力するというサイクルで成り立っている．

ところで，レコードを特定ボリューム上に書き出すまえに一定個数のレコードをグループ化するが，この過程を**ブロッキング**（blocking）と呼ぶ．ブロックとは，**レコード間ギャップ**（inter-record gap：略して IRG）の間に置かれたデータということができる．一般には**物理レコード**（physical record）という表現がとられる．つまり，磁気テープとか磁気ディスク上に物理的に記録されているレコードという意味である．これに対して，処理プログラムで取り扱われるレコードは，論理的に意味を持つ構成単位であることから**論理レコード**（logical record）と呼ばれ，物理レコードないしはブロック化レコードと区別される．ともあれ，ブロック化レコードは，1つ以上の論理レコードあるいはレコード・セグメントで構成される．複数個の論理レコードを1まとめにしてブロッキングすることによって，IRG の数が減少することからそれだけ外部記憶装置の無駄を排除できるのである．ブロッキングは単に記憶スペースの節約に貢献するだけではない．多くの場合，データ・セットの処理に必要な入出力オペレーションの回数が少なくなることから全体としての処理効率も高まるのである．

ところで，制御プログラムのもとでは，論理レコードとしてつぎの3つの形式が許される．固定長（F形式），可変長（V形式）および不定長（U形式）である．この3つの形式のどれを選択すべきかの主要なきめ手は，データ・セット自体の性質にかかっている．したがって，ユーザーは自分のプログラムで受

け取る入力の形式およびそのプログラムで書き出す出力の形式について知っている必要がある．この知識と，そのデータ・セットを収容するのに使われる入出力装置の種類およびデータ・レコードの読み書きに用いられるアクセス方式とにもとづいてレコード形式の選択を行うことになる．このレコード形式は，後述するように DCB マクロ命令，DD ステートメントないしはデータ・セット・ラベルのいずれかでの指定に従って，データ制御ブロックの特定フィールドに示される．つまり，レコード形式はデータ・セット属性として欠くことのできないものの１つなのである．

1. 固定長レコードの形式

固定長レコード（fixed length record：略してＦ形式）の長さは，その名で示されている通り，データ・セットを構成する全レコードとも同じ桁数である．Ｆ形式では，１個の論理レコードから成り立つ物理レコード，つまり，非ブロック化レコードも許されるし，複数個の論理レコードから成り立つ物理レコード，つまり，ブロック化レコードも許される（**図 6-2** 参照）．ただし，ブロック化レコードは磁気テープと DASD のデータ・セットだけに適用される．

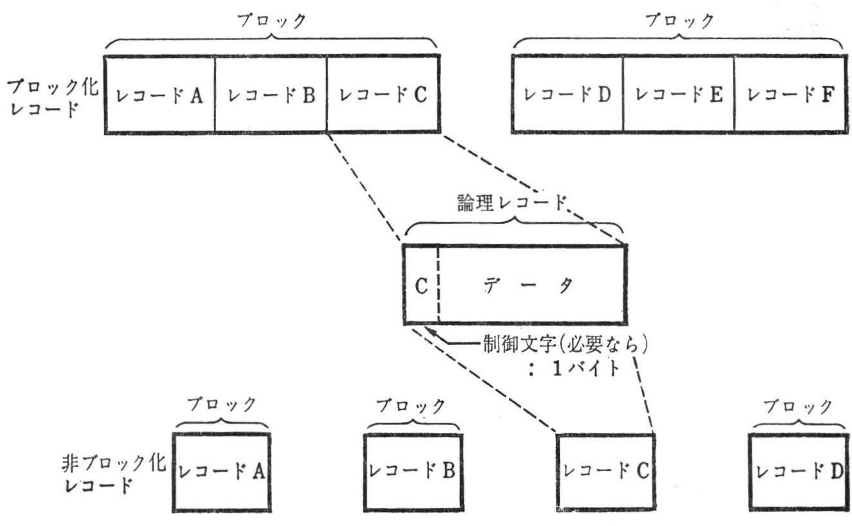


図 6-2 固定長レコード（Ｆ形式）

ブロック化された場合のブロック内のレコードの数はデータ・セット内の全ブロックともに同じであるが，最後のブロックだけは短いブロックとなる場合が考えられる．データ・セットを構成する全レコードをブロックを構成するレコード数（一般にこれを**ブロック化因数**というが）で除した場合，余りが生じるであろう．データ・セットの最後のブロックだけはこの余りの数のレコード数にならざるを得ないからである．

アクセス方式ルーチンでは，F形式のブロック化レコードに関しては自動的に長さの検査をしてくれる．ただし，前述の短いブロックについての考慮をしたうえでこの検査を行ってくれる．したがって，間違ったブロック長の物理レコードはシステムが検出してくれることになる．なお，長さの検査にはチャネル割込みの機能を用い，かつ，レコードのブロック化や非ブロック化の処理は，固定したレコードの桁数にもとづいて行うがゆえに後述する可変長レコードよりも速く処理することができるといえる．

図6-2で示しているようにF形式の論理レコードの最初の1バイトに制御文字を入れることができる．この制御文字は，印刷出力の紙送りの制御あるいは

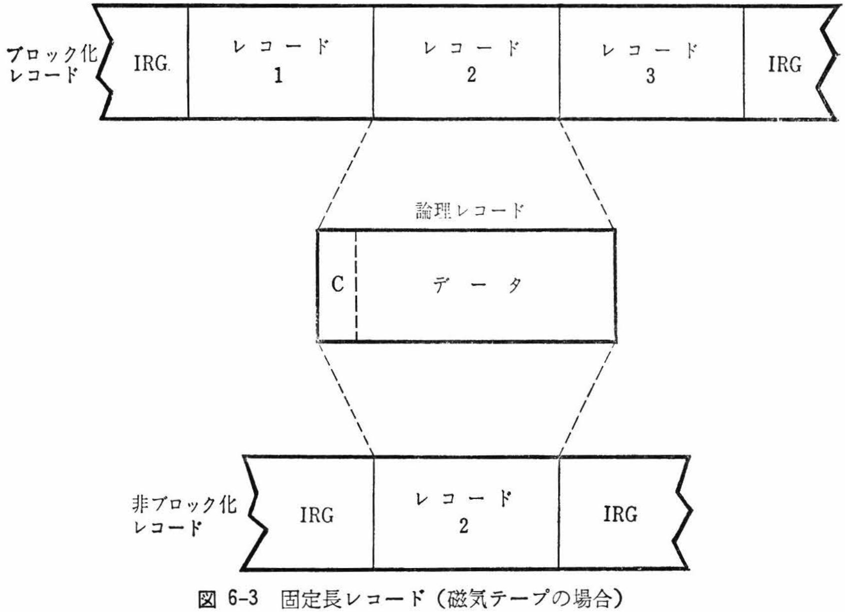


図 6-3　固定長レコード（磁気テープの場合）

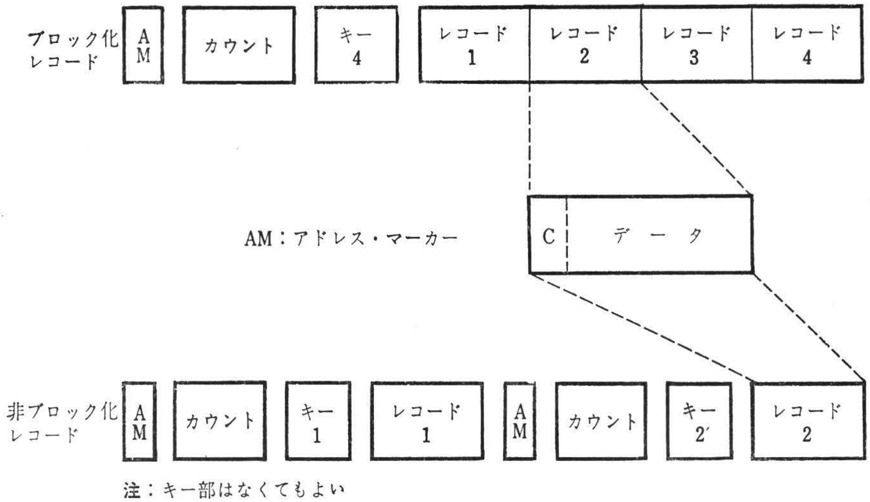


図 6-4　固定長レコード（DASDの場合）

穿孔出力レコードのスタッカーの選択のために用いられる文字である．各論理レコードの最初のバイトに制御文字が入っているかどうかは，DCB の所定のフィールドで示される．なお，この制御文字自体は印刷も穿孔もされないが，主記憶装置内ではレコードの一部をなしているので，この文字をも収容できる大きさのバッファ域を確保する必要がある．

図 6-3 と図 6-4 で磁気テープの場合の固定長レコードと DASD の場合の固定長レコードの様子を示しておいた．いずれの場合もブロック化レコード，非ブロック化レコードともに許される．DASD の場合には，キー部はあってもなくても差支えない．キー部をもった形式でかつブロック化レコードの場合には，キー部にはそのブロックの最後の論理レコードのキーが挿入されることになる．

2.　可変長レコードの形式

すべての論理レコードの桁数を一定に保つことができれば，プログラミング上は簡単になるし，処理効率もあがる．しかし，レコードの性質上固定長の形式をとることがむずかしい場合が生じる．このような場合には，可変長レコー

ド形式ないしは不定長レコード形式のいずれかを適用せざるを得ない．可変長レコードの形式では，各論理レコードの長さおよびブロックの長さをレコード自身の特定位置のフィールドで示す．一方，不定長の場合にはレコードの長さを示す何らの情報もレコード自身の中には持ち合わせない．

可変長レコード形式 (variable-length record) は，英字の頭文字をとって**V形式**という．V形式のレコードでは，論理レコードがブロックの長さを越える場合，論理レコードをブロック間にまたがらせることも許されている．このような形式のことを特に**分割可変長レコード** (spanned variable-length record) と呼び，普通の可変長レコード形式と区別する．分割可変長レコードは，**VS形式**ともいう．まず，V形式から説明しよう．

この形式は，各論理レコード自身の長さを持った可変長論理レコードと各ブロックの長さを持った可変長ブロックとから構成される．制御プログラムのデータ管理プログラムでは，この長さに関する情報を使ってブロック長の検査をしたり，ブロック化や非ブロック化の機能を果してくれるのである．

可変長レコードの一般形式を示したのが図6-5である．可変長レコードは，ブロック化レコードの形式も許されるし，非ブロック化形式のレコードも許される．したがって，可変長ブロックは，1つ以上の論理レコードから成り立つ．各ブロックの最初の4バイトは，ブロックの長さを記述する**ブロック記述語** (Block Descriptor Word：略して BDW) である．この最初の2バイトでブロックの長さ (LL) を指定する．ブロックの長さとは，BDW に要する4バイトにブロック内の全レコードの長さを加えたものである．BDW の3バイト目と4バイト目は，将来システムが利用することを考えて確保されており，ゼロにされている．データ管理プログラムでは，**待機アクセス技法** (queued accessing technique) を用いているときはブロック化を行ってくれるが，この場合にはデータ・セットを書き出すとき自動的にBDW にブロック長を挿入してくれる．ユーザー・プログラムで，ブロックの長さを算出し所定の個所に入れる必要はないのである．しかし，**基本アクセス技法** (basic accessing technique)を用いる場合には，データ管理プログラムではブロック化の機能を果してくれない．したがって，ブロックの長さは処理プログラムの中で算出し，BDW

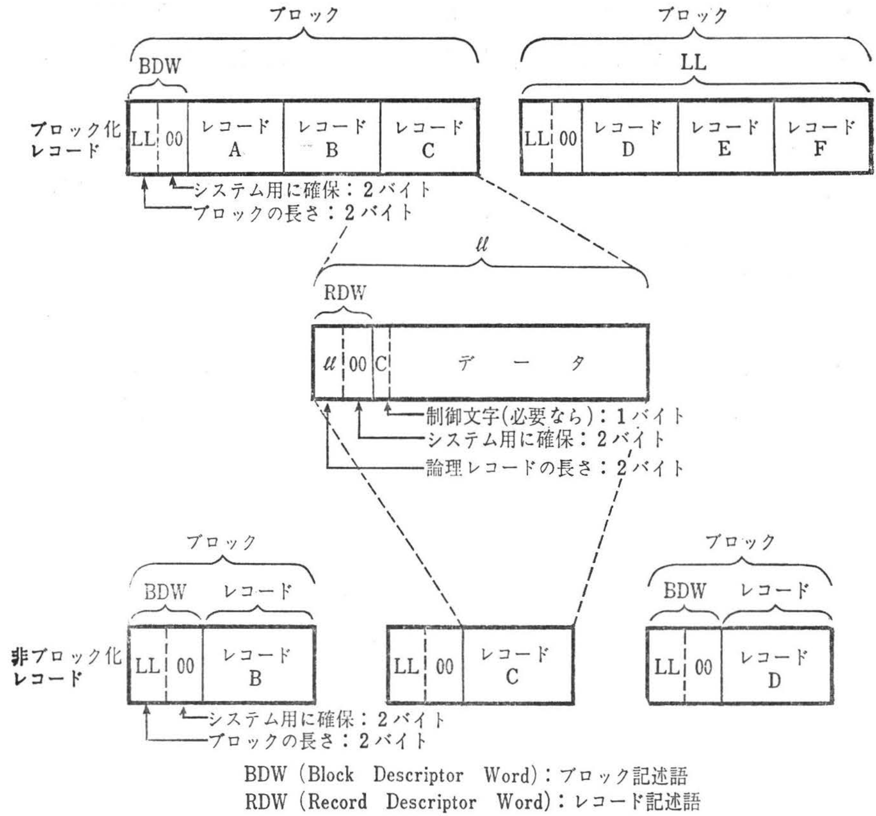


図 6-5 可変長レコード（V形式）

に入れる必要が生じるのである．

可変長ブロックを構成する可変長論理レコードは，レコード記述語とデータから成り立つ．**レコード記述語**（Record Descriptor Word：略して RDW）は，レコードの長さに関する情報を貯える4バイトのフィールドであり，レコードの直前に設けられる．この RDW の最初の2バイトで論理レコードの長さを示す．レコードの長さには，当然のことであるが RDW の4バイトも含まれる．なお，RDW の3バイト目と4バイト目は，V形式ではゼロである．データ管理プログラムでは，待機アクセス技法を使用しているかぎりブロック長は RDW に入っているレコード長をもとに自動的に算出してくれる．しかし，

データ管理プログラムといえども，新たに作成する論理レコードの個々の長さを知ることはできない．したがって，データ・セットの出力にさいして各レコードの長さをRDWに挿入するのはプログラマーの責任となる．逆に，データ・セットの入力にさいしては，処理すべき論理レコードの長さを制御プログラムの方で指示してくれる．すでにRDWに論理レコードの長さが記録されているので，この長さ情報を特定場所に入れてくれるのである．同時に長さ情報にもとづいて非ブロック化の機能を果してくれる．

図6-5でも示しているように，論理レコードの5番目のバイトには**制御文字**を入れることができる．制御文字は，固定長レコードの個所でふれたと同じ役割りを果す．なお，RDWも制御文字も印刷出力されたり穿孔出力されたりすることはない．一方，磁気テープや磁気ディスクが出力装置である場合にはそのまま書き出されることはいうまでもない．

V形式の可変長レコードに関して，磁気テープの場合とDASDの場合を示したのが図6-6と図6-7である．いずれの場合にも，ブロック化レコードと非ブロック化レコードの双方が許される．さらに，DASDの場合にはキー部を

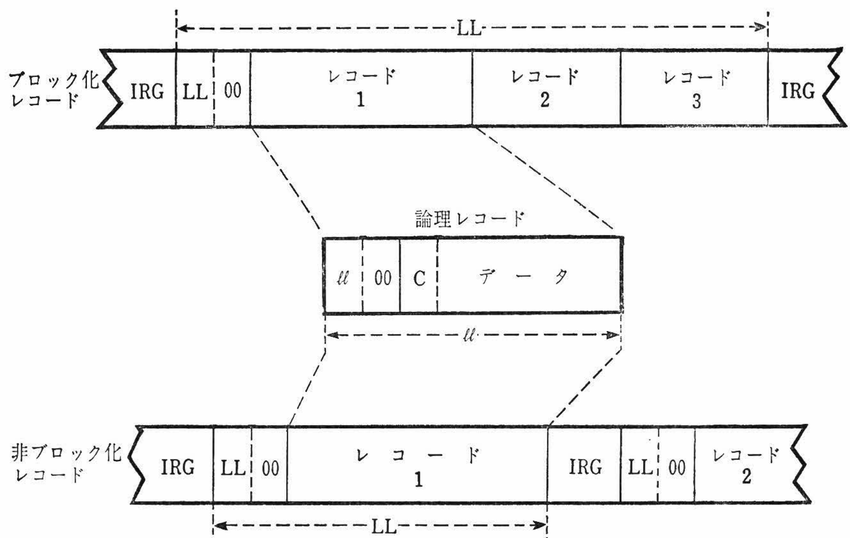


図6-6　可変長レコード（磁気テープの場合）

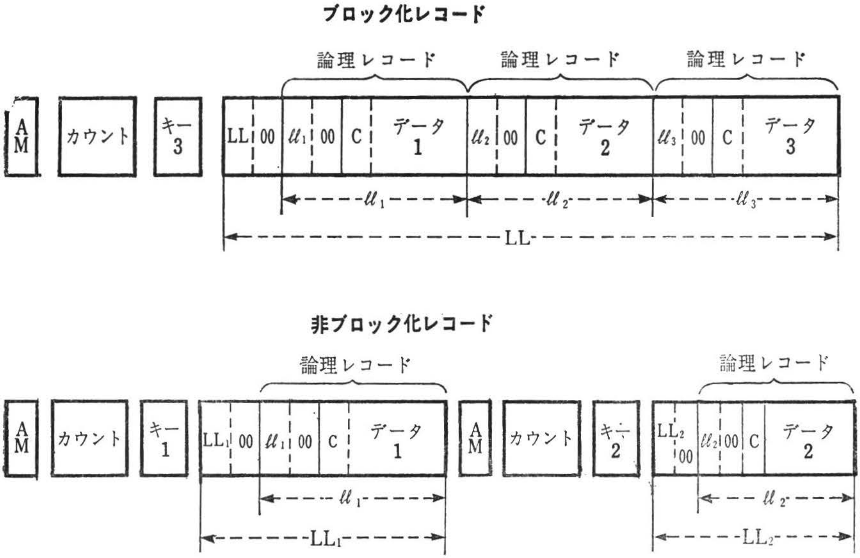


図 6-7　可変長レコード (DASDの場合)

持っていても，持っていなくてもどちらでもよい．ブロック化レコードでキー部を持っている場合には，そのブロック内の最後の論理レコードのキーがキー部に貯えられる．これは，固定長レコードの場合と同じである．

つぎに，分割可変長レコードの形式について説明しよう．

待機順次アクセス方式 (queued sequential access method) および**基本順次アクセス方式** (basic sequential access method) のレコード処理機能として，物理レコードの長さを越えるような可変長論理レコードを作成したり，処理することができるようになっている．つまり，V形式の場合だと論理レコードは，どんなに大きくても，物理レコードの長さを越えることは許されなかったが，この機能を適用すると論理レコードの長さの制約はなくなってしまう．論理レコードをセグメントに分割することによって，複数ブロックにレコードをまたがらせるのである．このような形式のことを，**分割可変長レコード**(spanned variable length record：略して VS 形式) という．この分割可変長レコードの形式を示したのが，**図 6-8** である．

ブロック化レコードを適用すると，データ管理プログラムでは各ブロックを

論理レコードないしはレコードのセグメントで満すようにする．一方，非ブロック化レコードの場合には**ブロック長**（block size）を越えるレコードは分割

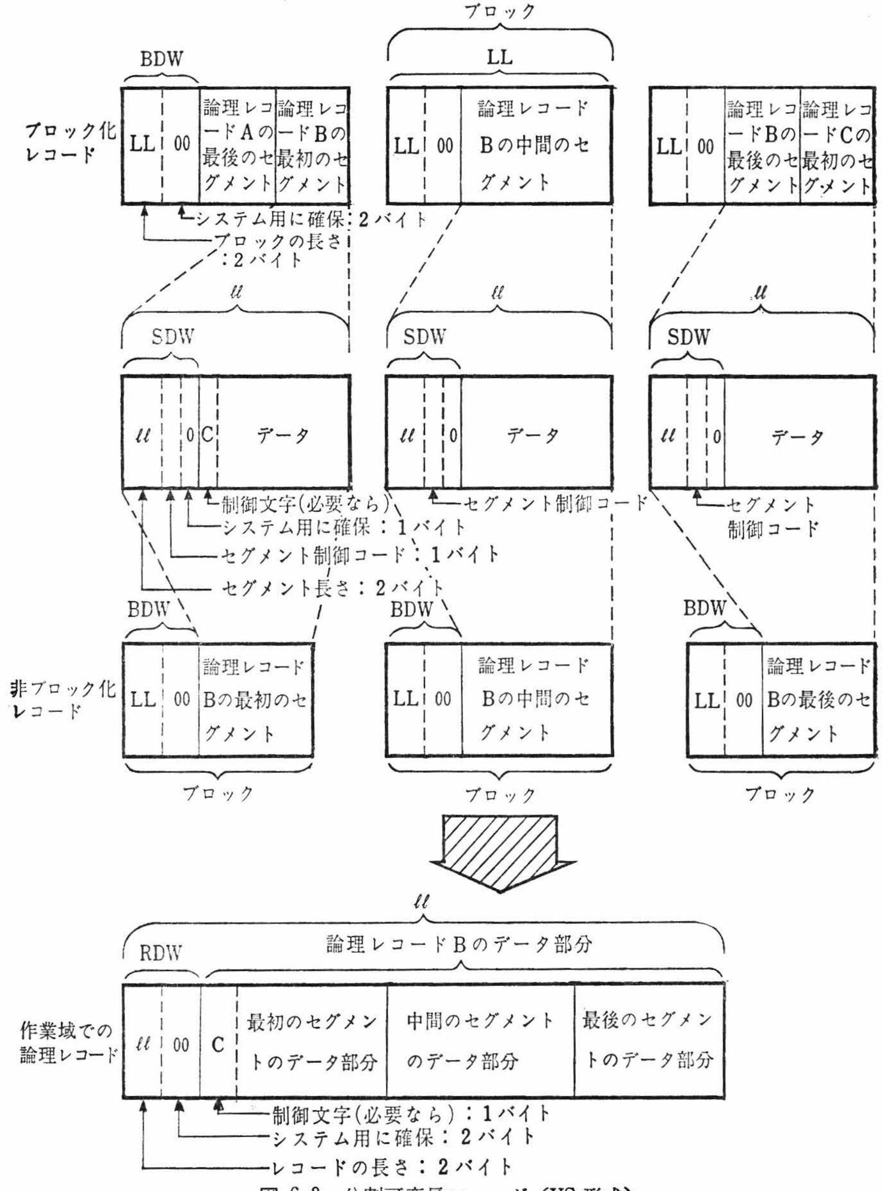


図 6-8　分割可変長レコード（VS 形式）

されてしまい，２つ以上のブロックにまたがって書き出される．したがって，各ブロックは，１個の論理レコードだけから構成されるか，１個のセグメントから構成されるかのどちらかである．図6-8の例では，非ブロック化レコードは，いずれもレコード・セグメントから構成されている．しかも，論理レコードが３つのセグメントに分割されて３つのブロックにまたがっている．逆に，ブロック化レコードの場合だと，各ブロックの大きさに見合うように別の論理レコードのセグメント（論理レコードそのものの場合もありうるが）と組合わせられてブロックが構成されることになる．

ところで，分割可変長レコードのブロック・サイズであるが，これも他の場合と同様にユーザーが適当な長さを指定する．したがって，使用する装置の種類，処理の形態，論理レコードの性質や大きさ，主記憶装置のうちでバッファとして許される大きさなどを勘案して最適な長さを設定する必要がある．ともあれ，分割可変長レコードではデータ・セットを構成するレコードのうちで最大の長さを持つレコードに全く拘束されない．つまり，論理レコードは，数ブロックにまたがってよい．時には複数個のボリュームにまたがっても一向に差し支えないのである．また，１つのブロックを論理レコードやレコード・セグメントの組合せで構成することもできる．ただし，１つのブロックが同一レコードの複数セグメントで構成されるということはあり得ないことである．同一ブロックに収容されるということは，同じレコードをそのブロック内でセグメントに分ける何らの理由もないからである．ともあれ，あるデータ・セットを形作る各レコードの長さに大きなバラツキがある場合にはV形式よりもVS形式を適用した方が効率的であるケースも少なくない．なお，データ・セットにレコードが追加されたり，逆に削除されたとき，あるいは異ったブロック・サイズやレコード・サイズで処理されたりすると，レコード・セグメントの長さは必然的に変わってしまう．

分割可変長レコードの場合にも，可変長レコード形式の場合と同様にブロック記述語（BDW）が存在する．その内容および機能は，可変長レコード形式の場合と全く同じである．一方，可変長レコード形式のレコード記述語（RDW）に相当するのが，**セグメント記述語**（segment descriptor word：略して

SDW）である．その形式は，図6-8の中で示している通りである．各レコード・セグメントはこの SDW とそれにひきつづいて存在するデータから成り立つ．SDW は，RDW と同じように4バイトのフィールドから構成され，そのセグメントに関する情報が貯えられる．最初の2バイトは，自らの SDW の4バイトをも含めたセグメントの長さが貯えられる．セグメントの長さは，その情報を貯える場所が2バイトであることの制約から，最大 32,756 バイトとなる．SDW の3番目のバイトが**セグメント制御コード**であり，この場所で論理レコード内でのセグメントの相対的位置が示される．実際には，セグメント制御コードは，このバイトの右側2ビットであり，そのコードの意味は**表6-2**の通りである．このコード以外のビットおよび4番目のバイトはすべてシステムが使用するために確保したものであり，ゼロ・ビットの状態である．

論理レコードの最初のセグメントの SDW だけは，そのレコードがセグメント化されたあとで RDW に置きかえられる．SDW をユーザが作るかデータ管理プログラムで作るかは，使用する処理方式により異る．BSAM では，分割レコードをユーザ・プログラムの中で作りあげ，処理する必要がある．一方，QSAM の**移動方式**（move mode）では，RDW を含めた完全な論理レコードがユーザの作業域で処理できる．これは，図6-8の最下端に示した通りである．GET マクロ命令を与えると，各セグメントを統合して完全な論理レコードにし，かつ RDW も作りあげて作業域に入れてくれるわけである．一方，論理レコードとそのレコード・サイズを RDW の所定位置に挿入したうえで PUT マクロ命令を与えると，必要なセグメントを形作り，かつ各セグメントの SDW を作りあげてくれる．

データ方式（data mode）は，分割可変長レコードだけに許される方式であ

2進コード	セグメントの相対的位置
00	論理レコードを完全に収容
01	複数セグメント・レコードの最初のセグメント
10	複数セグメント・レコードの最後のセグメント
11	複数セグメント・レコードの最初と最後以外のセグメント

表 6-2　セグメント制御コード

るが，レコードのデータ部分だけが移動するという点を除いて移動方式と全く同じである．つまり，データ方式ではユーザの作業域には論理レコードのデータ部分だけが移される．論理レコードの長さは，DCB の中の DCBLRECL フィールドでユーザに知らされる．移動方式では論理レコードの長さを含めて作業域で取扱えた点が異るだけである．

位置指定方式 (locate mode) では，GET マクロ命令，PUT マクロ命令のいずれでも一度に1つのセグメントだけを処理する．しかし，BUILDRCD マクロ命令でレコード域を指定するとか，DCB で BFTEK=A を指定してレコード域を要求することによって，論理レコード単位での処理も許されている．

このように QSAM においてはその処理方式により，分割可変長レコードの取扱いが若干異るのである．

以上順次アクセス方式における分割可変長レコード形式の特徴について説明してきたが，これまでの説明を要約するために具体例を示したのが**図 6-9** である．ブロック化レコードの場合と非ブロック化形式の場合とでセグメンテーションの仕方が異ることがこの例から容易に理解できよう．ブロック化レコードの場合だと，各ブロックが一ぱいになるようにレコードのセグメンテーションを行ってゆく．一方，非ブロック化レコードでは，各ブロックには1つの論理レコードかセグメントが収容できるようなブロックを作りあげてゆくのである．

なお，分割可変長レコードの機能は**基本直接アクセス方式** (basic direct access method：略して BDAM) にも適用される．BDAM においては，非ブロック化レコードだけしか取扱えない．つまり，BDAM において分割可変長レコードの機能を用いると1トラックを越えるような可変長非ブロック化の論理レコードを作成したり，処理したりすることができるようになる．また，レコードを分割してセグメントにすることによって，トラックに可変長レコードを詰め込むこともできる．

BDAM に対して分割した非ブロック化レコードの形式を指定し，かつ論理レコードが完全に1つのトラックに収容できない場合には，レコード・セグメントでそのトラックを満そうとする．それゆえに，データ・セットを構成する最大のレコードの長さがブロック・サイズによって何らの制約もされないので

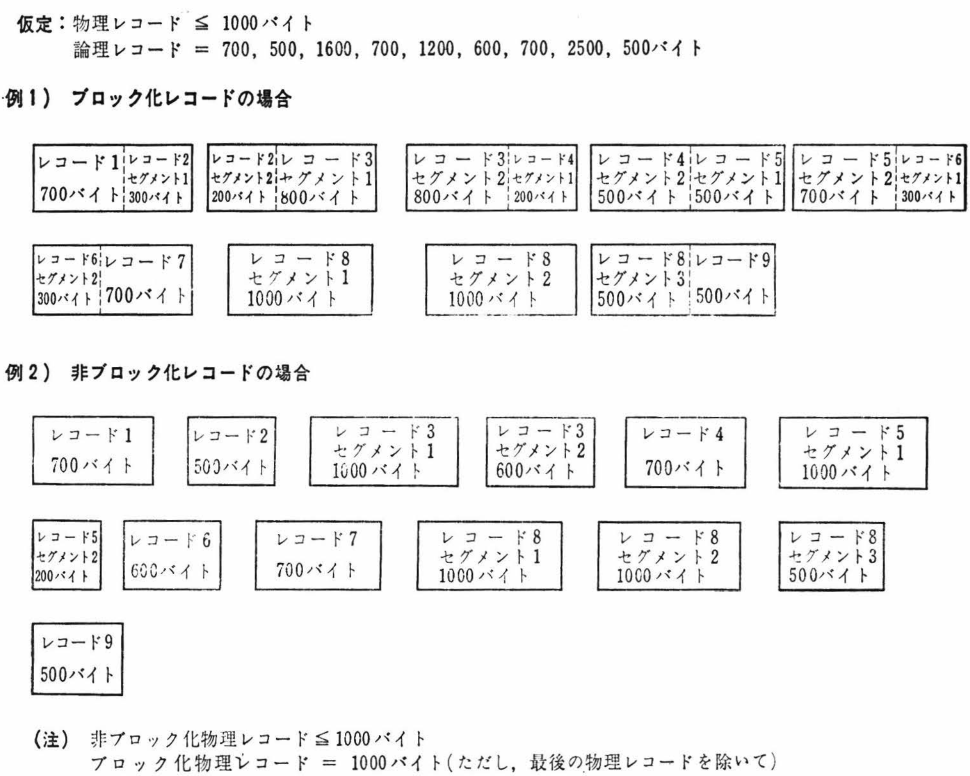


図 6-9　分割可変長レコードの例

ある．レコードの各セグメントが異ったトラックに収容されることによって，レコードが数トラックにまたがることが許されるからである．しかし，BDAMの場合には，レコードがボリューム間にまたがることは許されない．したがって，ある論理レコードを構成するすべてのセグメントが同一ボリュームにおさめられることが必要である．

3. 不定長レコードの形式

3番目の形式は，**不定長レコード** (undefined record) の形式である．この

形式では非ブロック化レコードしか許されない．もし，ブロック化レコードをこの形式で扱うとしたら，ブロッキング，ディブロッキングの処理はすべてユーザ・プログラムで行うことになる．

不定長レコードは，本質的には可変長レコードである．しかし，可変長レコードと異るのは，レコード自身の中に，何ら長さに関する情報を持ち合わせないことである．その形式を示したのが図 6-10である．各ブロックとも非ブロック化レコードとして取扱われるのであるから，レコードがブロック化されている場合には自分のプログラムでディブロッキングを行うよりほかないのである．図でも示したように，制御文字を各レコードの最初のバイトに挿入して用いることは許される．不定長レコードの形式では，各レコードの長さをユーザ・プログラムの中で算出してシステムに教える必要性がある．システム，つまり，データ管理プログラムでは，指示された長さにもとづいてブロックの書き出しを行う．逆に，不定長レコードの読み込みにさいしては読み込んだブロックの長さをシステムがユーザ・プログラムに知らせてくれる．ユーザ・プログラムでは示された長さにもとづいて該当レコードの処理を行うことになる．いずれにしろ，システムとしては不定長レコードの長さのチェックを行うことはできないので，ユーザ・プログラムの方でそれだけ十分な注意を払うことが要求されることとなる．

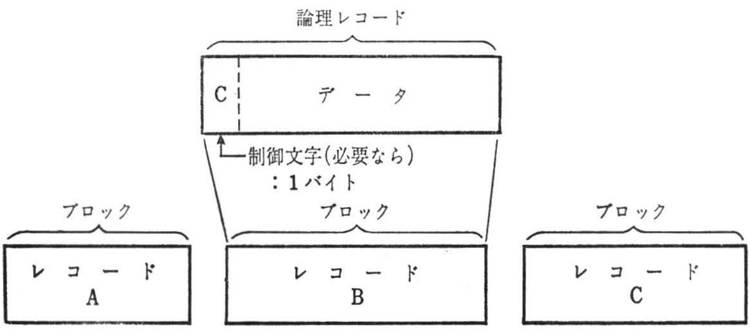


図 6-10　不定形式レコード（U形式）

ともあれ，オペレーティング・システムのデータ管理機能で取り扱うことのできるレコードの形式としては，固定長レコードの形式（F形式），可変長レコードの形式（V形式とVS形式）および不定長レコードの形式（U形式）である．固定長レコードの形式だと処理効率もあがり，処理プログラム自体の負担も少なくてすむ．可変長や不定長レコードの形式だと若干処理効率もさがるし，プログラム上わずらわしさが残る．しかし，どのレコード形式を採用するかは，本来的にレコードに持たされている性質やレコードの処理の仕方などにかかっている．どの形式を採用するにしろ，データ管理プログラムの機能としてはその入出力オペレーションのためのかなりの面倒をみてくれるわけである．このようなレコードの形式は，このあと説明するデータ・セットの編成法やアクセスの方式とも大きなかかわりをもつ．したがってこの章の主旨であるデータ・セットの編成法とアクセス方式を詳述するに先立って，まずレコードの形式について簡単に説明したのである．

三. 待機アクセス技法と基本アクセス技法

オペレーティング・システムのもとでは，データ管理ルーチンで読み書きするレコードの処理にプログラマーが出来るだけ専念できるような配慮がなされていることはいうまでもない．入出力オペレーションに関してプログラマーに要求される主な責任は，処理すべきデータ・セットの諸特性を記述することであり，使用する**バッファ割当て手法**（buffering technique）と**アクセス方式**（access method）とを記述することである．アクセス方式は第5章の三節で説明したようにデータ・セットの編成法とデータを処理するのに用いるアクセス技法の組合わせによって決まる．データ管理プログラムで一般に取り扱うことのできるデータ・セットの編成法には，順次編成，索引順次編成，直接編成および区分編成の4種類がある．一方，**データのアクセス技法**（data access technique），つまりデータ・セット内のレコードの読み書きの仕方には2種類の方法がある．すなわち，待ち行列を使った読み書きと待ち行列を使わない読み書きである．前者を**待機アクセス技法**（queued access technique），後者を

基本アクセス技法（basic access technique）と呼んでいる．両者の間では，入出力バッファの使い方や入出力オペレーションと CPU 処理との同期のとり方が異る．ともあれ，データ・セットの編成法とレコードの読み書きの仕方には密接な関係があることから，この２つの組合わせでアクセス方法を定めているのである．どのアクセス方法でレコードの処理を行うかを決めることは重要である．レコードの処理法の決め方によって，処理効率がかなり違ってくるからである．アクセス方式については後で詳しく説明することにして，まず待機アクセス技法と基本アクセス技法について説明しよう．

1. 待機アクセス技法

待機アクセス技法（queued access technique）では，GET と PUT マクロ命令を用いて主記憶装置と補助記憶装置との間でのデータ転送を行う．これらのマクロ命令を用いると，補助記憶装置へレコードを記録するさいのブロッキングおよびレコードの取り出しのさいのディブロッキングを自動的に行ってくれる．また，バッファにレコードを先き読みすることおよび入出力オペレーションと CPU 処理とをオーバーラップ（同期化）することも，待機アクセス技法の自動的な機能である．

バッファ処理をオペレーティング・システムのデータ管理プログラムで制御してくれるがゆえに，プログラマーは必要なだけの入出力バッファを活用することができる．しかも，バッファにレコードを出し入れするのにそのつど GET または PUT マクロ命令を出す必要はない．入力の場合，バッファ内のすべての論理レコードが処理されると自動的に次のブロックを読み込んでくれる．逆に出力の場合には，バッファが論理レコードで満されると自動的に書き出しを開始してくれる．したがって，通常２つ以上のブロックが主記憶装置に貯えられているので，入出力オペレーションのために論理レコードの処理が待たされることはない．このような方法で，入出力オペレーションと CPU 処理とが最大限同期化できるのが待機アクセス技法である．

入出力オペレーションと CPU 処理との同期化は制御プログラムで行ってくれるので，入出力オペレーションが完了したか，入出力エラーが生じたかある

いは例外状態が生じたかのテストをユーザー・プログラムで行う必要はない．GET または PUT マクロ命令を与えたあと入力バッファが物理レコードで満されるか，出力バッファの物理レコードが出力されて使用可能状態になるまでは，制御権はユーザー・プログラムの所定の個所に戻されるということはない．それゆえに，プログラマーは，何時でも GET マクロ命令を与えたあとは論理レコードが処理可能な状態にある積りでプログラムをコーディングしてゆけばよい．また出力の場合には，PUT マクロ命令を与えたあとは出力バッファに論理レコードの組立てが出来る状態を想定してプログラミングしてゆけばよいことになる．なお，入出力エラーの分析（SYNAD），ボリュームの終りあるいはデータ・セットの終り（EODAD）の処理ルーチンへのブランチも必要なとき自動的に行われる．

1）GET マクロ命令……レコードの取り出し

GET マクロ命令は，入力データ・セットからレコードを取り出してくれる．この命令は論理的には順次処理で，かつ装置から独立した方法で機能してくれる．GET マクロ命令は必要に応じて入力バッファに物理レコードを読み込み，物理レコードをディブロックし，また入力エラーが生じた場合の**エラー回復手順**（error recovery procedure）の実行を指示する．

順次データ・セットでかつレコード形式が分割可変長レコードの場合だと，レコード・セグメントを組み合わせて論理レコードを仕立てあげてくれる．順次データ・セットのレコードをすべて処理し，データ・セットの終りを示す標識をキャッチすると自動的にデータ・セット・ラベルを検査したうえで，制御権をユーザーが指定したデータの終り（EODAD）ルーチンに移す．順次データ・セットで複数ボリュームにまたがっている場合にボリュームの終りの状態が発見されると**自動的にボリュームの切り換え**（automatic volume switching）を行ってくれる．これが GET マクロ命令の主要な機能である．

2）PUT マクロ命令……レコードの書き出し

PUT マクロ命令は物理レコードを出力データ・セット上に書き出す．GET

マクロ命令と同様に，この命令も論理的には順次処理であり，かつ装置から独立した方法で作動する．PUT マクロ命令は必要に応じて，出力バッファの物理レコードの書き出しを行い，論理レコードをブロッキングし，また出力エラーが生じた場合にはエラー訂正の手順を実行してくれる．さらに順次データ・セットに対しては，ボリュームの切換えやラベルの作成を自動的に開始してくれる．分割可変長レコードの場合だと，論理レコードの分割も行ってくれる．

カード穿孔装置ないしは印刷装置に対して PUT マクロ命令が与えられるとF形式またはV形式のブロック当りのレコード数を自動的に1に調整する．この機能があるので，レコードを直接印刷ないしは穿孔しないで一時的に磁気テープ・ボリュームか直接アクセス・ボリュームに貯えおく場合，ブロックの大きさ（BLKSIZE）を最適の値にしておくことができるのである．なお，分割可変長レコードに対するブロックの大きさはカード1枚の穿孔行あるいは1行の印刷行に等しい値にしておく必要がある．したがって，この場合にはレコードの大きさ（LRECL）の方がブロックの大きさ（BLKSIZE）よりも大きい値を指定することができる．

3) PUTX マクロ命令……更新レコードの書き出し

PUTX マクロ命令は，入力データ・セットのレコードをもとにデータ・セットの更新を行ったり，出力データ・セットを作り出したりするのに使われる．PUTX マクロ命令では，既存のデータ・セットのレコードを更新したり，置き換えたり，挿入したりすることができる．しかし，PUT マクロ命令のような別のデータ・セットのレコードを作りあげたり，レコードの追加を行う機能はない。

PUTX マクロ命令でレコードの更新を行う場合，各々のレコードは直前に与えられた GET マクロ命令で指定されたと同じデータ・セットに書きもどされる．つまり，更新レコードが収容されているバッファには目印し（flag）がつけられており，直接アクセス記憶装置上の読み出されたと同じ場所に書き戻されるのである．そのブロックは PUTX マクロ命令が実行されたときに書き出されるのではない．次のバッファに対して GET マクロ命令が与えられたと

き，はじめて書き戻されるのである．

入力データ・セットにもとづいて出力データ・セットを作り出すのにPUTXマクロ命令を用いているときに，PUT マクロ命令を使ってそのデータ・セットに新しいレコードを追加することもできる．PUTXマクロ命令は，必要に応じてレコードをブロッキングし，出力バッファの書き出しをスケジュールし，エラーが生じた場合の出力エラー訂正手順を実行する機能を持っている．

2. 基本アクセス技法

基本アクセス技法 (basic access technique) では，主記憶装置と補助記憶装置との間でデータ転送を行うのに READ マクロ命令と WRITE マクロ命令を用いる．この技法を用いるのは，オペレーティング・システムがレコードの処理順序を予測できない場合ないしは待機アクセス技法が行う自動的な機能のいくつかあるいはそのすべてを使いたくない場合である．この技法では，待機アクセス技法のような先読みの機能や後書き出しの機能はもたないし，CPU処理と同期化された入出力オペレーションのスケジューリング機能もない．しかし，これらの機能をプログラミングするのに都合のよいマクロ命令は用意されている．このマクロ命令を用いてユーザー・プログラムの中にこれら機能を組み込むことが許されているのである．

原則として READ および WRITE マクロ命令は論理レコード単位での処理は行ってくれない．ブロック単位の処理だけである．つまり，READ マクロ命令を実行するつど該当バッファに新たな物理レコードが持ち込まれる．物理レコードを構成する論理レコードがいくつあろうが全く無関心である．逆に WRITE マクロ命令が実行されるつど，出力バッファにあるデータが1ブロックとして書き出される．出力バッファに論理レコードを何個どういう順序で詰め込むかはすべてユーザー・プログラムの責任である．したがって，レコードのブロッキングだとかディブロッキングはユーザーが行う必要がある．

さらに付言すると，READ および WRITE マクロ命令は単に入出力オペレーションを開始させるだけである．それゆえ，入出力オペレーションが正しく完了したかどうかを確認するために CHECK マクロ命令か WAIT マクロ命令

を使ってテストする必要がある．これらのマクロ命令を使って入出力オペレーションの完了をテストすると同時にその同期化を計ってゆくのである．

1) READ マクロ命令……ブロックの読み取り

READマクロ命令は，入力データ・セットからデータ・ブロックを取り出して，主記憶装置の指定された場所に入れてくれる．入力オペレーションとCPU処理とをオーバラップさせるために，入力オペレーションが開始されると制御権はユーザー・プログラムに戻される．ユーザー・プログラムでは目下の入力オペレーションの影響をうけない処理をオーバラップさせる．入力オペレーションで得られたレコードを処理するためには，その読み取りオペレーションが正しく完了したかどうかを確認するために，読み取りオペレーションのために作られた DECB (Data Event Control Block) の内容をテストする必要がある．

DECB は，各々の READ および WRITE マクロ命令で確保される16バイトから32バイトの記憶域である．この制御ブロックには入出力オペレーションに必要な各種の制御情報（イベント制御ブロック，入出力オペレーションの形式，ブロックの長さ，DCB のアドレス，バッファのアドレスなど）が貯えられる．入出力オペレーションが終了すると，この DECB をチェック・ルーチンで検査し，入出力エラーや例外事態が発生しているかどうかを決定する．これのために使われるのが DECB である．

索引順次データ・セットを読み取ると，ブロックは主記憶装置のバッファに読み込まれ，必要な論理レコードのアドレスが DECB の所定のフィールドで示される方法がとられている．

ともあれ，READ マクロ命令では，処理すべきデータ・セットの編成法とシステムが行う処理タイプに応じていろいろな入力オペレーションの形式が指定できるようになっている．それぞれの形式は READ マクロ命令のオペランドで記号を用いて指示する．順次編成の場合だと，SF（データ・セットを順次的に読み取る）と SB（磁気テープ・データ・セットを逆方向に読み取る）とがある．索引順次編成だと，K（データ・セットを普通に読み取る）と KU

（更新のためにデータ・セットを読み取る．システムは直接アクセス装置上のレコードのアドレスを自動的に保存してくれる．したがって，更新したレコードを WRITE マクロ命令で書き戻すさいにはこのアドレスを使えばよい）とがある．直接編成の場合には，D（直接アクセス方式を用いる），I（ブロック標識を用いて，該当ブロックをつきとめる），K（キーを用いてブロックをつきとめる），F（直接アクセス装置の記憶位置をフィードバックする），X（ブロックを排他的に制御する）などの記号を組み合わせて種々の処理タイプを指示できる．

2) WRITE マクロ命令……ブロックの書き出し

WRITE マクロ命令は，主記憶装置の指定された記憶域から出力データ・セットにデータ・ブロックを書き出す．WRITE マクロ命令の別の機能として，更新レコードをデータ・セットの元の場所に書き戻すことにも使われる．出力オペレーションと CPU 処理とをオーバラップさせるために，書き出しオペレーションが開始されると制御権はユーザー・プログラムに返される．ユーザー・プログラムでは目下の出力オペレーションに関係のない処理を出力オペレーションに併行させて続行するようにする．次の出力オペレーションの準備を行うためには（正確には DECB を再使用する前に），書き出しオペレーションが正しく完了したかどうかを確認するために，その書き出しオペレーションのために作られた DECB をテストする必要がある．

READ マクロ命令と同様に， WRITE マクロ命令にも処理すべきデータ・セットの編成法とシステムが行う処理タイプとに応じた種々の出力オペレーションの形式が指定可能になっている．それぞれの処理形式は READ マクロ命令の場合と同じように WRITE マクロ命令のオペランドで指定する．

順次編成のデータ・セットだと，SF（順次的にデータ・セットを書き出す）と SFR（次に出力すべき DASD のアドレスをフィードバックしながら，順次的にデータ・セットを書き出す）の2種類がある．

索引順次編成のデータ・セットの場合には，K（更新されたレコードを含んだブロックの書き出し，または同じキーを持った非ブロック化レコード同志の

置き換え）か KN（新しいレコードの書き出し，または可変長レコードの長さの変更）かの指示ができる.

直接編成データ・セットだと，SD（見せかけの固定長レコードの書き出し）と SZ（トラック記述レコード，つまりレコード0の書き出し．システムは必要なデータを与え，レコード0を書き出したあとで次のトラックに進む）とがある．さらに通常のオペレーションを行うためのものとして，D（直接アクセス方式を用いる），I（探索引数がブロックを示す），K（探索引数がキーである），A（新しいブロックを追加する），F（レコードの記憶位置を示す情報をフィードバックする），X（排他的制御を解除する）などの記号を組み合わせて種々の処理タイプが指定できる.

3) CHECK マクロ命令……入出力オペレーションの完了テスト

基本アクセス技法を使ってデータ・セットを処理している場合，入出力要求が終るのを待ち，正しく終了したかどうかをテストするために CHECK マクロ命令を用いる．システムは Data Event Control Block（DECB）を用いてエラーや例外事態の発生をテストする．同じデータ・セットにこのマクロ命令を続けて出す場合には，対応する READ または WRITE マクロ命令と同じ順序で出す必要がある.

チェック・ルーチンで入出力エラーを発見すると，データ制御ブロック（DCB）で指定されたエラー分析のための**出口ルーチン**（exit routine, 一般に **SYNAD ルーチン**という）に制御権が移される．そこでエラーの回復処理を行う． DCB で出口ルーチンが指定されていない場合には，異常状態とみなしてタスクの実行を終了させてしまう．また，順次データ・セットを入力としている場合にデータの終りであることが発見されると，DCB で指示された**出口ルーチン**（EODAD : End-Of-Data Address）に制御権が移される．この EODAD は，入力データ・セットの最終処理を行うためのユーザー・ルーチンのアドレスである．このルーチンには，前述したように入力要求が出されて，しかもすでに取り出すべきレコードやブロックが存在しない場合に自動的に制御権が移されるようになっている．ユーザー・ルーチンでは，データ・セットを CL-

OSE したり，別の順次データ・セットを処理したりすることができるのである．もし，DCB で EODAD が指示されていない場合には，異常事態とみなしてタスクの実行を終了させてしまうことになる．なお，ボリュームの終りをキャッチした場合には，自動的に**ボリュームの切り換え**（volume switching）の手順をとってくれる．

4）WAIT マクロ命令……入出力オペレーションの完了待ち

データ・セットの処理を行っている場合，READ ないしは WRITE オペレーションの完了をテストするのに WAIT マクロ命令を使うことができる．この命令を用いることによって，入出力オペレーションと CPU 処理との同期化をはかることができる．しかし，WAIT マクロ命令には CHECK マクロ命令のような DECB を検査して入出力エラーや例外事態を検出する機能やボリュームの自動的切り換えの機能は持ち合わせていない．したがって，WAIT マクロ命令を用いる場合にはこのような機能を果すためのルーチンをユーザー・プログラムの中に組み入れておく必要があることは言うまでもない．

なお，このマクロ命令では複数個の READ/WRITE オペレーションの完了を待つのに用いることもできる．この場合には，各々のオペレーションを別々にテストないしは検査する必要がある．

これまで説明してきた待機アクセス技法と基本的アクセス技法が，データ・セットの編成法との関連で具体的にどのような方法で利用されるかを以下の章で検討してゆくこととしよう．

四．順次データ・セットの処理

順次編成のデータ・セットでは，レコードは物理的な順番に並んでいる．したがって，次のレコードの位置は前のレコードの位置から自動的に決定される．どんな入出力装置でも順次データ・セットとして取扱える．また，直接アクセス・ボリューム以外のすべてのボリュームのデータ・セットは順次データ・セットとしてしか処理できない．更に，直接アクセス・ボリューム上にある

データ・セットは，その編成法に関係なく順次的に処理することも可能である．データ管理プログラムにはこのような機能があるので，順次データ・セットではプログラムを実行するときに使用する入出力装置のタイプをほとんど考慮しないでコーディングすることもできるのである．つまり，装置から独立したプログラムが書けるのである．もちろん，装置から独立したプログラムを書くためには，装置に依存したある種のマクロ命令や処理機能を用いることに対する制約があるのも事実である．

順次データ・セットの読み書きには，待機アクセス技法と基本アクセス技法のどちらも使える．さらに，**連鎖スケジューリング**（chained scheduling）と呼ばれる手法を用いると，順次データ・セットに必要な入出力オペレーションの効率を高めることができる．通常，順次データ・セットは，GET，PUT マクロ命令で入出力する．つまり，待機アクセス技法を使用する．しかし，データ・セットがブロッキングされていなくてプログラムがさほど複雑にならないような場合だと，READ，WRITE マクロ命令を使用した方がよい．READ，WRITE マクロ命令を使うことによって処理速度を高めることが可能だからである．

ここで，前述の**連鎖スケジューリング**（chained scheduling）について言及しておこう．連鎖スケジューリングを要求すると，データ管理プログラムは通常の入出力ルーチンを使用する代りに，数個の入出力オペレーションをダイナミックに結合して実行してくれる．連鎖スケジュリーングでは，一連の独立した別個の入出力オペレーションを１つの連続したオペレーションとして実行するようにチャネルに要求を出す．この場合，入出力オペレーションの同期化を行うために，CCW の中の **PCI**（program-controlled interruption）**標識**が用いられる．このような方法がとられると，入出力開始のための CPU 命令の実行回数とチャネルのスタート・ストップの回数も少なくなる．その結果，主記憶装置と補助記憶装置間のデータ転送に要する CPU 時間およびチャネルのスタート・ストップ時間の両方が削減されることになり，全体としての入出力オペレーションの効率は向上する．特に，DASD の場合の回転待ち時間は大幅に減少できる．というのは，別々に要求された複数個の連続する物理レコード

を一回の回転で取り出すことができるからである．もし，連鎖スケジューリングを用いないと，1つの物理レコードを取り出すごとに1回転要するのであるから，その差は無視できない．これまでの説明から推察されるように連鎖スケジューリングを適用するデータ・セットには少なくとも2つ以上のバッファが割り当てられていないと意味をなさない．

なお，データ制御ブロックが OPEN されたとき，つぎのような条件がある場合には連鎖スケジューリングの要求は無視され，通常のスケジューリングが行われる：

- トラックあふれ機能
- OPEN マクロ命令のオペランドで更新（UPDAT）が指定されている
- 交換バッファ割り当て
- CNTRL マクロ命令の使用
- 入出力装置のタイプが紙テープである

連鎖スケジューリングは，広範囲の入出力オペレーションを必要とするプログラムの場合には非常に重要な意味をもつ．連鎖スケジューリングを適用するデータ・セットがチャネルの利用可能時間の大半を占有してしまい，他のデータ・セットの入出力オペレーションが入り込む余地がなくなってしまうからである．したがって，2つ以上のデータ・セットを処理する場合には可能なかぎり別々のチャネルを割り当てるように配慮することが必要になる．

説明をもとにもどそう．順次データ・セットでは待機アクセス技法と基本アクセス技法の両方が適用できる．順次データ・セットと待機アクセス技法を組み合わせたアクセス方式が，**QSAM** (Queued Sequential Access Method) である．一方，順次データ・セットと基本アクセス技法を組み合わせたアクセス方式は，**BSAM** (Basic Sequential Access Method) である．そこで，QSAM と BSAM の具体的な使用法について概観してゆこう．

1. QSAM による順次データ・セットの処理

第5章の「九．バッファの確保と制御機能」で説明したように，待機アクセス技法では，主記憶装置内でのレコードの処理方式として3種類の方法が選択

可能である．つまり，**位置指定方式**（locate mode），**移動方式**（move mode）および**置換方式**（substitute mode）である．これらの処理方式のどれを用いるかは後述するように，DCB マクロ命令の MACRF フィールドで指定する．なお，位置指定方式の GET マクロ命令と組み合せて PUTX マクロ命令が使用できる．このマクロ命令を用いると，更新されたレコードをもとのデータ・セットに書きもどすことができる．QSAM によるデータ・セットの処理の仕方として，以上の4つの場合を例にとりあげて説明することにする．

1） 移動方式による処理

単純バッファ割り当てによる移動方式の例を示したのが，**図6-11**である．問題は極めて簡単にしてある．入力データ・セットの論理レコードを何ら加工することなく，そのまま出力している．ただし，出力データ・セットの物理レコードはブロッキングされている．ブロック化因数は3である．一方，入力データ・セットの物理レコードは非ブロック化レコードである．

データ・セットの処理を行わせるためには，データ・セットの属性（データ・セットの編成法，レコードの形式，レコードの長さ，ブロックサイズなど），処理法の記述（使用する入出力マクロ命令の形式，入出力バッファに関する情

```
                 :
①           OPEN   (INDCB, (INPUT), OUTDCB, (OUTPUT))
② YOMU      GET    INDCB, INWORK
③           MVC    OUTWORK, INWORK
④           PUT    OUTDCB, OUTWORK
            B      YOMU
⑤ OWARI     CLOSE  (INDCB,, OUTDCB)
                 :
⑥ INWORK    DS     CL80
⑦ OUTWORK   DS     CL80
⑧ INDCB     DCB    DSORG=PS, MACRF=(GM), DDNAME=INDD,      C
                   RECFM=F, BLKSIZE=80, EODAD=OWARI
⑨ OUTDCB    DCB    DSORG=PS, MACRF=(PM), DDNAME=OUTDD,     C
                   RECFM=FB, BLKSIZE=240, LRECL=80
                 :
```

図 6-11　順次データ・セットの処理－移動方式

報など），エグジット情報（データの終りになったときのエグジット・ルーチンのアドレス，エラー分析のためのエグジット・ルーチンのアドレスなど）を与えておく必要がある．これらの情報をまとめて収容する記憶域は，いうまでもなく**データ制御ブロック**（DCB：Data Control Block）である．DCB はそれぞれのデータ・セットに対応して必要であり，DCB マクロ命令を用いてプログラムの中に作り出す(図6-11⑧と⑨参照)．図6-11において，入力データ・セットのための DCB には INDCB という名前が，出力データ・セットの DCB には OUTDCB という名前がつけられ，それぞれ DCB マクロ命令を用いて必要な情報が定義されている．データ・セットの読み取りや書き出しを行うときには DCB を参照する必要があるが，その場合には GET，PUT マクロ命令のオペランドでこれら名前を用いるようにする．なお，DCB の詳細については，第5章の「八．データ制御ブロック（DCB）の機能と構成」を参照されたい．

DCB マクロ命令は，図6-11で分るようにオペランドはいくつかのキーワード・パラメータから構成される．キーワード・パラメータの記入順序は任意であり，パラメータとパラメータの間はカンマ（，）で区切って続ける．1行に収容しきれないときは，カラム72に継続行の指定をして次の行に続ければよい．DCB マクロ命令で必ず指定する必要のあるキーワード・パラメータは DSORG と MACRF の2つだけである．その他のパラメータの情報は，プログラムの実行時に DD ステートメントで指定するか，データ・セット・ラベルから得ることができる．すでに述べたように，プログラムにできるだけ柔軟性を持たせるために，処理プログラムのコーディングにさいしては変動しない情報だけを指定して，変動する情報はできるだけプログラムの実行時に指定するようにする．しかも，オペレーティング・システムではそれが十分許されるように設計されているのである．図6-11の例においても，できるだけ一般的なプログラムとするために主要なパラメータしか指定していない．残りのパラメータは実行時に指定するか，ないしは省略時解釈をとらせるようにしている．**省略時解釈**とは，パラメータを省略してしまったとき，自動的にとられる値である．たとえば，図6-11の DCB マクロ命令では，バッファ・プールの自動設定

に関するパラメータ BUFNO, BUFL あるいは BFALN のいずれも指定していない．これらのパラメータがDDステートメントでも指定されていないとするとつぎのような値がとられる：

- BUFNO＝2：2個のバッファを確保
- BUFL＝80（出力データ・セットの場合は240）：BLKSIZE で指定した値が使われる
- BFALN＝D：バッファの境界合せは倍ワード

これが省略時解釈の例である．なお，バッファの自動設定に関しては，第5章の「九．バッファの確保と制御機能」を参照されたい．

図 6-11の例で用いている DCB マクロ命令のパラメータの意味については，表 6-3 にまとめてある．

図6-11の⑧の DCB マクロ命令をみていただきたい． DSORG＝PS と指定されているのは，Physical Sequential（物理的順次編成），つまり順次編成のデータ・セットが使われることを意味している．⑧では LRECL のパラメータが省略されている．固定長レコード（RECFM＝F）で，非ブロック化の場合は論理レコードの長さ（LRECL）がブロックサイズ（BLKSIZE）に等しいので省略できるのである．このデータ・セットは入力データ・セットであるので，入力レコードが最後まで処理されたとき制御権を移すアドレスを EODAD＝OWARI で示している．出力データ・セットの場合には，このパラメータは不要である．なお，DCB と DD ステートメントとの対応づけを行うために DDNAME パラメータを指定する． したがって，この例のプログラムを実行するさいのジョブ制御カードの DD ステートメントの1つ，つまり入力データ・セットに対応する DD ステートメントの名前欄には必ず INDD と指定して関係づけを行うのである．

⑨の DCB マクロ命令は，出力データ・セットのためのものである．出力データ・セットのレコードは，固定長ブロック化レコード（RECFM＝FB）で各ブロックの長さは240バイト（BLKSIZE＝240）で，論理レコードは80バイト（LRECL＝80）であることを示している．したがって，ブロック化因数は3で

パラメータ	摘　　要
DSORG=PS *	Data Set Organization=Physical Sequential の意味であり，データ・セットの編成法が物理的順次編成であることの指定.
MACRF={(GM)/(PM)} *	Macro Form={GET Move mode/PUT Move mode} の意味で，入出力のマクロ命令の形式として GET, PUT マクロ命令を使用し，かつ移動方式で処理することを指定.
BLKSIZE=	Block Size の意味で，物理レコード（ブロック）の長さをバイト単位で指定.
LRECL=	Logical Record Length の意味で，論理レコードの長さをバイト単位で指定.
RECFM={F / FB / V / VB / VS / VBS / U}	Record Format（レコード形式）を指定. F は Fixed：固定長レコード FB は Fixed Blocked：固定長ブロック化レコード. V は Variable：可変長レコード VB は Variable Blocked：可変長ブロック化レコード. VS は Variable Spanned：可変長分割レコード. VBS は Variable Blocked Spanned：可変長ブロック化分割レコード. U は Undefined：不定長レコード.
EODAD=	End of Data Set Address の意味. 入力データ・セットの場合だけ指定する．入力レコードのすべてを処理し終ったときにブランチすべきユーザ・ルーチンの記号名を指定.
DDNAME=	Data Definition Name（データ定義名）を英字で始まる8桁以内の英数字で指定.

＊のパラメータは DCB マクロで必ず指定，他は省略可能.

表 6-3　DCB マクロ命令の主要なパラメータ

ある．なお，データ定義名（DDNAME）は CUTDD としている．

移動方式の説明に移るまえに，OPEN, CLOSE マクロ命令に関して若干説明しておこう．

プログラム中に作った DCB とデータ・セットを結びつけ，データ・セットを使用可能にするのが OPEN マクロ命令である．その働きを図示したのが図

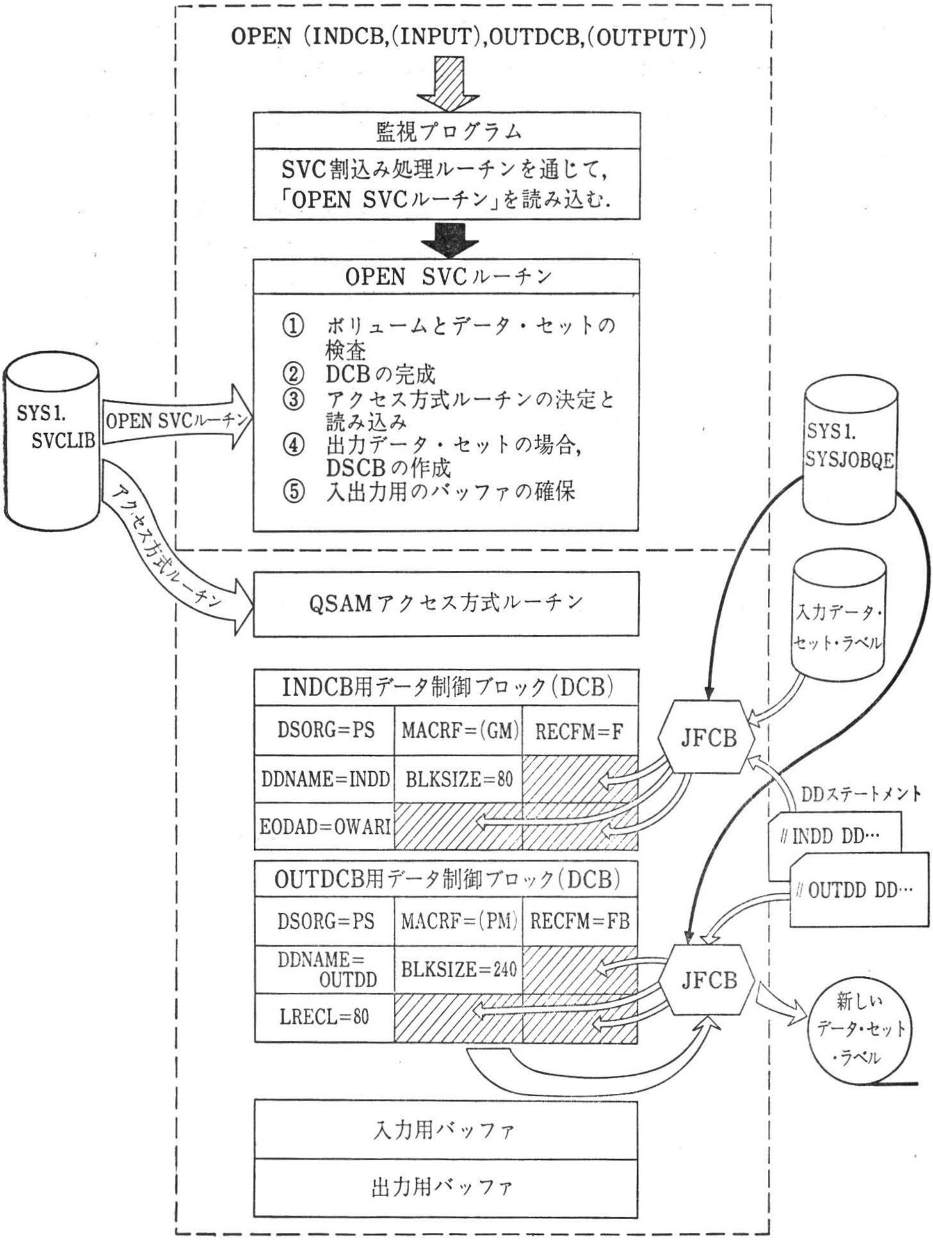


図 6-12 OPEN マクロ命令の働き

6-12である．DCB のフィールドを構成する情報源は，DCB マクロ命令のほかに DD ステートメントとデータ・セット・ラベルがある．「OPEN SVC ルーチン」が実行されると DCB の各フィールドはすべて完成される．出力データ・セットの場合だとデータ・セット・ラベルを作成し書き出す．DCB の内容から，データ・セットの編成法，バッファ割り当て方法，アクセス方式および入出力装置の特性を判定し，それにふさわしいアクセス方式ルーチンを選択し，主記憶装置にロードする．さらには，DCB のバッファに関する情報にもとづいてバッファを確保し，データ・セットの処理開始のための準備を完了する．

図6-11の①が OPEN マクロ命令の書き方である．オペランドはカンマで区切ることによって，一度に複数個のデータ・セットを OPEN できる．DCB のアドレスとカッコで囲んだオプション（INPUT または OUTPUT）とが対になって，1個のデータ・セットの情報を表わす．ただし，INPUT の場合にはそのオプションを省略しカンマで省略したことを示すことも許される．

データ・セットの処理が終了したら，その後始末をしデータ・セットを処理プログラムから切り離すために CLOSE マクロ命令を与える．CLOSE マクロ命令の働きを示したのが図6-13である．出力データ・セットの場合には，バッファにまだ残っているレコードをこの時点で最後の物理レコードとして出力する．さらに，出力データ・セットのトレーラ・ラベルの作成ないしは更新を行って書き出す．入力の場合だとトレーラ・ラベルの検査を行う．そのあとで，DD ステートメントないしは CLOSE マクロ命令で指示された後始末をボリュームおよびデータ・セットに対して実施する．そのうえ，CLOSE された DCB は，OPEN される前の状態に戻される．こうしておくことによって，同一 DCB を別のデータ・セットの処理に適用することが可能となる．また，そのデータ・セットのために設けられていた各種制御ブロックやテーブル，バッファあるいはアクセス方式ルーチンで占められていた記憶域を他の目的で利用できるように解放する．これによって，処理プログラムとデータ・セットの関係は完全に切り離されてしまうのである．

図6-11の⑤が CLOSE マクロ命令の書き方である．OPEN マクロ命令の場合と同様一度に複数個のデータ・セットが CLOSE できる．また，DCB のア

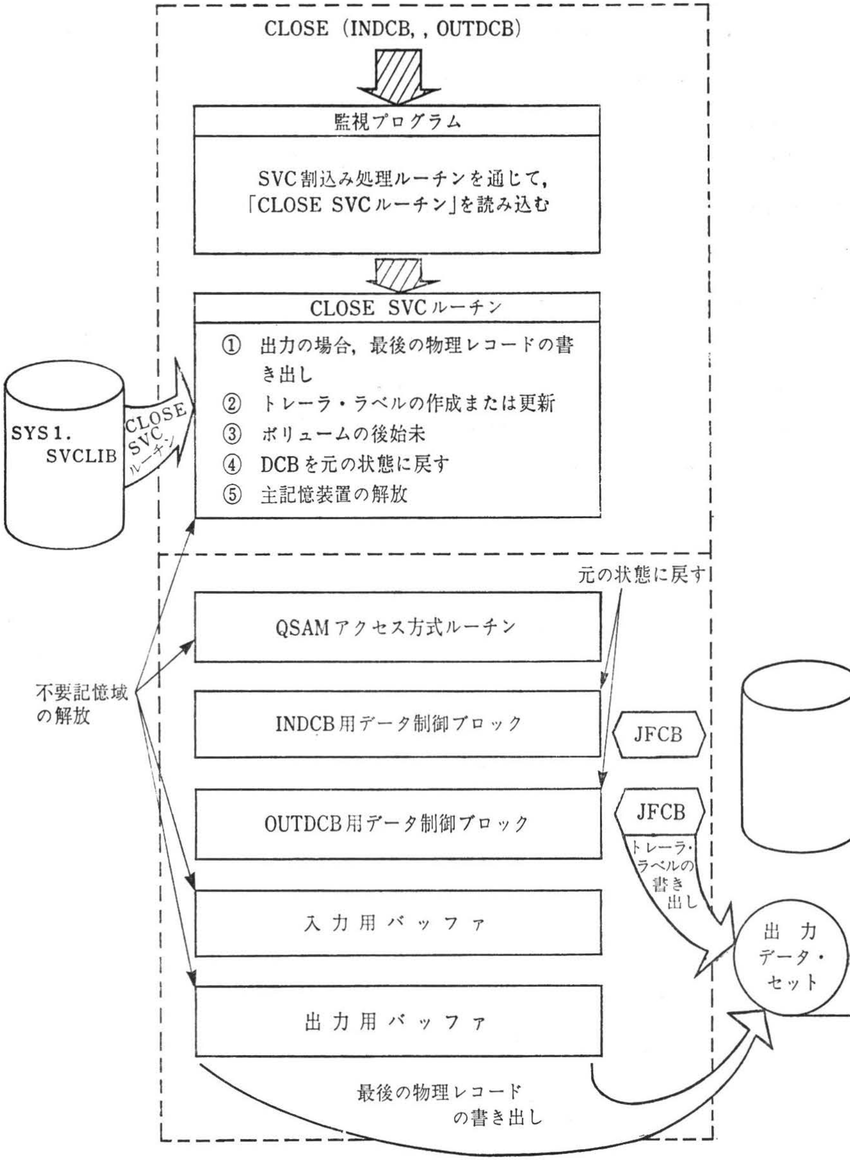


図 6-13 CLOSE マクロ命令の働き

ドレスとオプションの2個のオペランドを対にして1個のデータ・セットの情報を表わす．⑤ではオプションは INDCB, OUTDCB ともに省略されている．したがって，INDCB の直後にカンマを2個続けてオプションの省略を示している．OUTDCB のあとには特にカンマは不要である．CLOSE マクロ命令のオプションでは磁気テープまたは直接アクセス装置のボリュームとデータ・セットの後始末の仕方の指示ができる．しかし，これらの指示はプログラムの実行段階に DD ステートメントを通じて行うことができるので，CLOSE マクロ命令では通常省略する．

なお，OPEN と CLOSE の詳細については第5章の「十．データ・セットの OPEN と CLOSE 機能」を参照されたい．

さて，いよいよ移動方式による順次データ・セットの処理の仕方について説

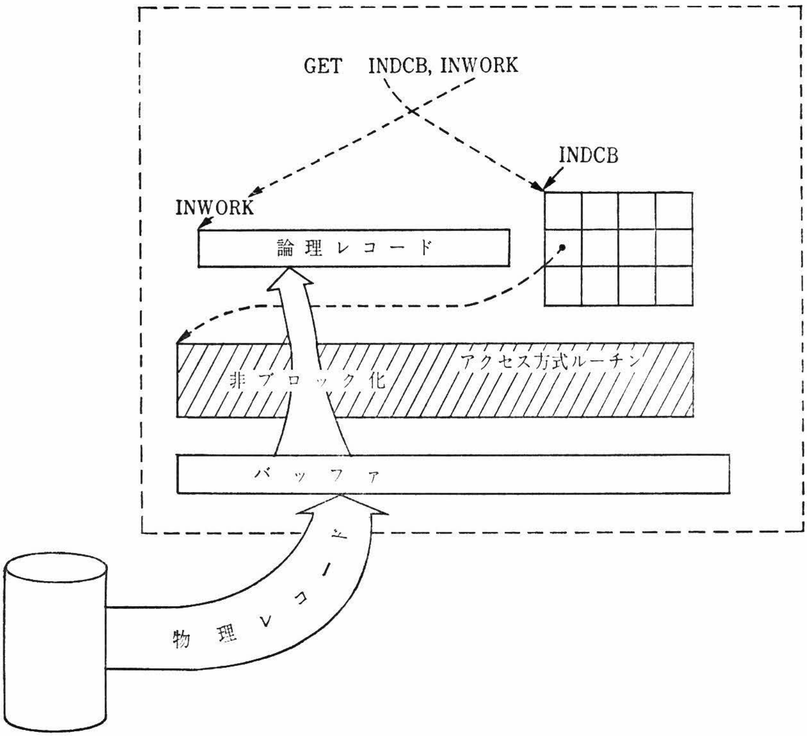


図 6-14　GET マクロ命令（移動方式）の働き

明しよう．これが本項目での主な主旨である．これ以前に説明したことはすべてのアクセス方式に共通することであって，特に移動方式特有のことではない．説明の都合上，この部分で取扱ったに過ぎない．

図6-11を参照しながら移動方式の処理の仕方を見ていこう．②が移動方式のデータ・セットの読み取り命令である．この GET 命令により，データ・セットから物理レコードをバッファに読み込む．非ブロック化レコードであれば，そのまま作業域にそのレコードが移されるが，ブロック化されている場合にはアクセス方式ルーチンでディブロッキングして作業域に移す（図6-14参照）．このようにバッファからいったん作業域にレコードが移される方式が**移動方式**（move mode）である．移動方式の GET の場合には，DCB マクロ命令で MACRF=(GM)，すなわち GET マクロ命令による Move mode である旨の指定が必要である．そのうえで，GET マクロ命令の第2オペランドで作業域のアドレスを指示することが要求される．

ここで，ディブロッキングの過程を少し詳しく説明しよう．図6-15をみていただきたい．最初の GET 命令で物理レコードが読み込まれ，最初の論理レコードが作業域に移る（①）．2番目の GET ではバッファの2番目の論理レコードが作業域に移り（②），3番目の GET で3番目の論理レコードが移る（③）．この例では，ブロック化因数3でブロック化されているので，3番目の論理レコードを作業域に移すことによって，バッファは次の物理レコードを読み込んでも一向に差支えなくなる．したがって，3番目の GET でバッファの最後の論理レコードが作業域に移されると自動的に次の物理レコードの読み込みを開始する．つまり，3番目の論理レコードの処理と2番目の物理レコードの読み込みとがオーバーラップされるのである．4番目の GET では，2番目の物理レコードが読み込まれ，入出力エラーや例外事態のチェックが完全に終ったことが確認されたあとで，4番目の論理レコードが作業域に移される(④)．このようなステップがくり返し行われるのである．その結果，GET 命令が実行されるつど，作業域にはいつでも論理レコードが入っている．

なお，図6-15ではバッファが1個の場合を示している．しかし，普通は2個以上のバッファを割り当てる．この場合には，一方のバッファに読込みが終る

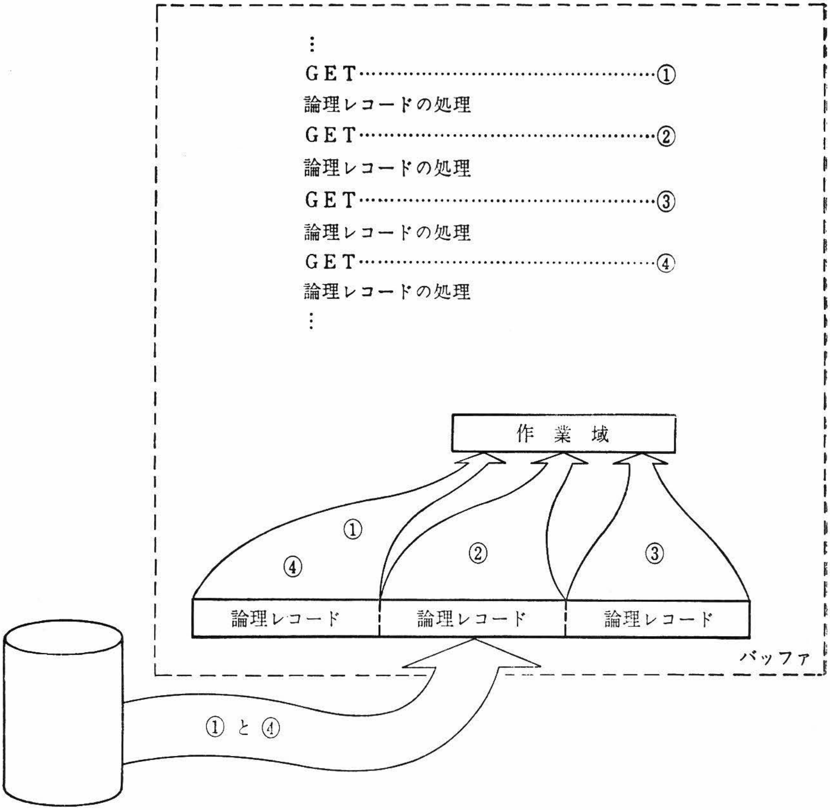


図 6-15　GET マクロ命令（移動方式）のディブロッキング機能

とすぐに別のバッファへの読み込みを開始してくれる．すべてのバッファが物理レコードで満されるまでどんどん先に読み取っていってくれる．処理が終ったバッファが出来るとそこにも読み込んでいく．このようにして，処理と入出力オペレーションを最大限にオーバーラップさせてくれるのが待機アクセス技法の特徴である．

再度図 6-11の②を参照していただきたい．オペランドには，入力データ・セットの DCB アドレスと作業域のアドレスを記入する．それぞれのアドレスは②で示しているように記号名で示してもよいし，別の方法としてはアドレスをあらかじめレジスターにロードしておき，そのレジスター番号をカッコで囲ん

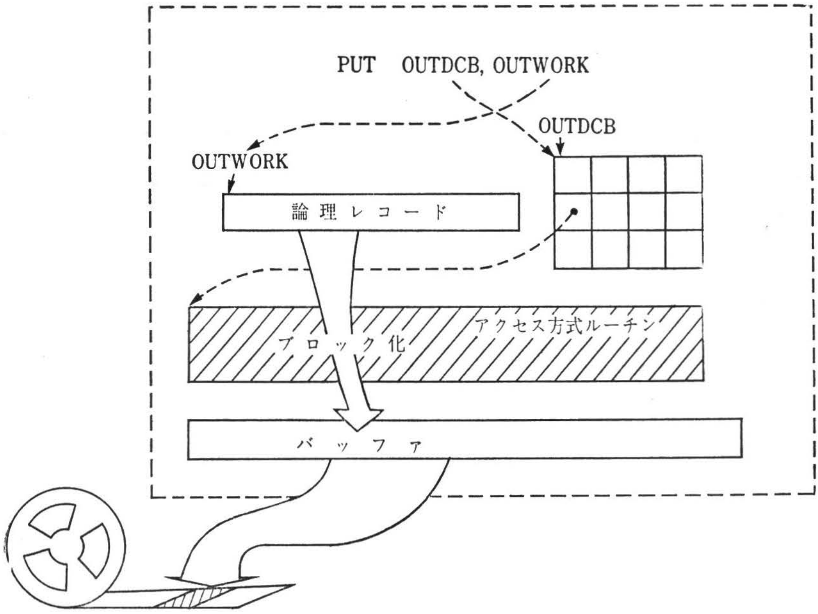


図 6-16 PUT マクロ命令（移動方式）の働き

で指定する方法もある．作業域は論理レコードの大きさに合せた記憶域をユーザー・プログラムの中で確保しておく必要のあることは言うまでもない（図6-11⑥）．

移動方式の書き出しも，基本的には読み取りの場合と同じ考えである．ただデータの流れが逆になるだけである．つまり，PUT マクロ命令により，作業域の論理レコードがバッファに移される．そして，RECFM＝FB の指定があればブロッキングを行い，物理レコードとして出力装置へ書き出す（図 6-16参照）．

移動方式の PUT マクロ命令を使うためには，DCB で MACRF＝(PM) と指定する必要があり，かつ PUT マクロ命令のオペランドで作業域のアドレスを指示することが要求される．

さて，PUT マクロ命令によってブロッキングがどのように達成されるかみておこう．ブロック化因数 3 の場合の処理を図 6-17に示してあるので参照されたい．GET マクロ命令の場合とは逆の流れになる．1番目と2番目の PUT

で作業域にある論理レコードがバッファに移り，３番目の PUT では３番目の論理レコードが作業域に移ることによって物理レコードが出来上るので，これを書き出す．４番目の PUT では，バッファの最初のレコードとして論理レコードが移る．このようなくり返しで，ブロック化されたレコードが出力されてゆく．

この例でもバッファが１個の場合であるが，バッファが２個以上設定されているとあるバッファからの書き出しを行っている最中に別の空いているバッファにどんどん作業域から論理レコードを移してゆく．出来上った物理レコードは前の物理レコードが書き出されたあと書き出される．これが，後書き出しである．このような方法で，出力オペレーションと処理とを最大限オーバーラップしてくれる．

図6-11の④が PUT マクロ命令の書き方の例である．GET の場合と同様，

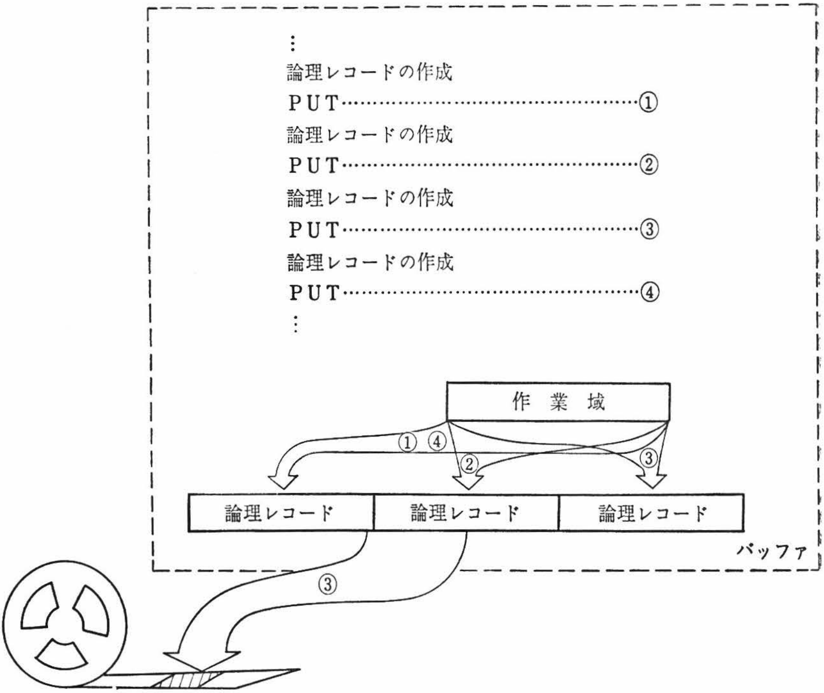


図 6-17　PUTマクロ命令（移動方式）のブロッキング機能

DCB のアドレスと作業域のアドレスを指定する．アドレスは，例のように記号名であってもよいし，アドレスがロードされたレジスター番号をカッコでくくって示したものであってもよい．

図 6-11の例では，入力データ・セットと出力データ・セットに別々の作業域を確保している．しかし，両者に共通の作業域を使っても差支えない．その場合には，入力作業域から出力作業域にレコードを移動するための③の命令は不要である．共通の作業域の記号名を WORK とすると，この部分はつぎのようになる：

```
YOMU   GET INDCB, WORK
       PUT OUTDCB, WORK
```

なお，この例では入力データ・セットも出力データ・セットも移動方式を使っているが，移動方式と位置指定方式の組合せも自由に選択して使うことができる．

2) 位置指定方式による処理

位置指定方式（locate mode）は，移動方式と異なり作業域は使用しない．位置指定方式の GET，PUT マクロ命令ではデータの移動を伴なわない．入

```
          ⋮
        OPEN   (INDCB, (INPUT), OUTDCB, (OUTPUT))
LOOP    PUT    OUTDCB
        LR     6, 1
        GET    INDCB
        MVC    0 (80, 6), 0(1)
        B      LOOP
OWARI   CLOSE  (INDCB,, OUTDCB)
          ⋮
INDCB   DCB    DSORG=PS, MACRF=(GL), DDNAME=INDD,        C
               RECFM=F, BLKSZE=80, EODAD=OWARI
OUTDCB  DCB    DSORG=PS, MACRF=(PL), DDNAME=OUTDD,       C
               RECFM=FB, BLKSIZE=240, LRECL=80
          ⋮
```

図 6-18　順次データ・セットの処理 − 位置指定方式

力バッファないしは出力バッファ内でそのまま処理を行う方式がとられる．移動方式では，必要な論理レコードは作業域に移されるか，出力の場合だと作業域から出力バッファに移された．しかし，位置指定方式はその名前で示す通りバッファの所要のレコードの位置をポインターで示してくれる．**ポインター**とは，処理の対象となるレコードあるいは次に出力すべき論理レコードの入るべきバッファ内のアドレスをレジスターで示してくれる方法である．

単純バッファ割り当てによる位置指定方式の例を示したのが**図 6-18**である．問題は，移動方式に使ったものと全く同じである．入力データ・セットは非ブロック化の固定長レコードからなり，各レコードは80バイトである．この入力データ・セットをそのまま出力し，ブロック化因数3の出力データ・セットを作りあげる．図 6-11の移動方式の場合のプログラムと比較してみていただきたい．

まず，DCB マクロ命令の差はどこに表われているであろう．使用するマクロ命令の形式を指定する場所，つまり MACRF のパラメータだけが異っているであろう．入力データ・セットの場合には，MACRF=(GM) が MACRF=(GL) となっている．GET Locate mode のマクロ命令の形式を使うということである．出力データ・セットの場合も，MACRF=(PM) が MACRF=(PL) となっている．PUT Locate mode の意味である．

当然のことながら，GET と PUT マクロ命令の形式も異る．作業域は不要であるからマクロ命令のオペランドでは DCB のアドレスを指定するだけでよい．例では記号名でアドレスを示しているが，あらかじめ DCB のアドレスをレジスターへロードし，そのレジスター番号を指定する方法であってもよい．

さて，位置指定方式においてレジスター1がどのような役割りを果しているかを**図 6-19**で理解してもらおう．この図では，入力データ・セットのブロック化因数を2，出力データ・セットのそれを3と仮定している．

最初の GET で物理レコードが読み込まれ，最初の論理レコードのアドレスがレジスター1に入れられる．次の GET ではバッファ内の2番目の論理レコードのアドレスがレジスター1に入る．3番目の GET では新しい物理レコードがバッファに読み込まれたうえで，バッファ内の最初の論理レコードがレジ

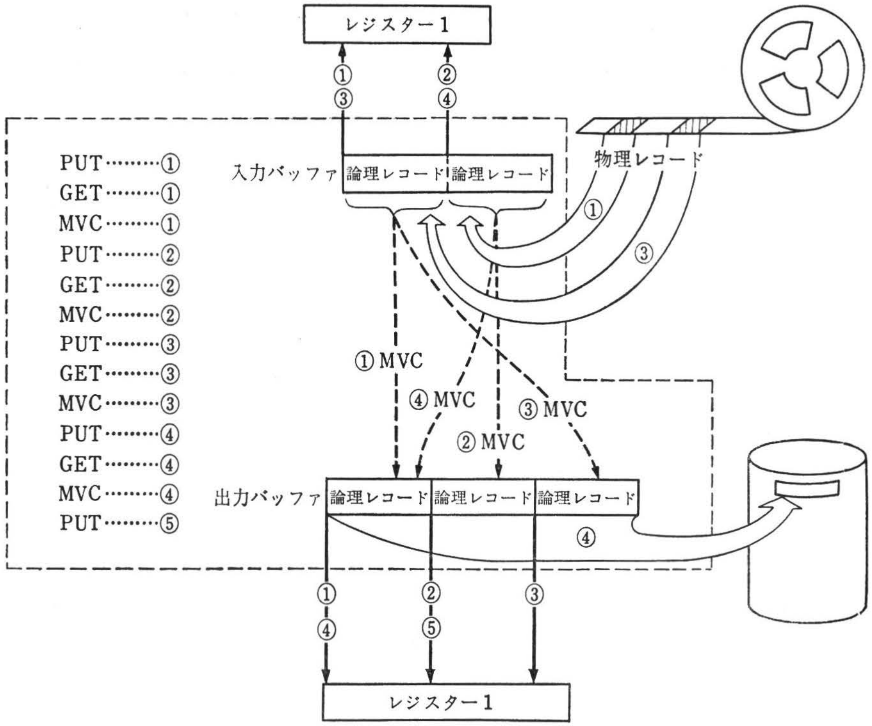


図 6-19　位置指定方式におけるレジスター1の役割り

スター1に入る．つまり，GET マクロ命令が実行されるつど，次に処理すべき論理レコードのアドレスがレジスター1に入れられている．したがって，レジスター1のアドレスを基点にすることによって，論理レコード内のどのフィールドでも参照し得るのである．

一方，PUT マクロ命令の場合にもレジスター1を使用する．最初の PUT で出力バッファの最初のアドレスがレジスター1に入る．レジスター1のアドレスを使って出力すべき論理レコードを出力バッファに移動できよう．次のPUT ではバッファ内の次の論理レコードを入れるべき位置のアドレスがレジスター1に入る．3番目の PUT では次の位置のアドレスが，そして4番目のPUT ではバッファ内の物理レコードを出力したうえで，出力バッファの最初のアドレスがレジスター1に入る．PUT マクロ命令が実行されると，常に次に

使用可能な出力バッファのアドレスがレジスター1に入れられるのである．

このように，レジスター1で示す位置を変更してゆくことによって，レコードのブロッキング，ディブロッキングを達成してくれるのである．なお，**図6-19**では入力と出力いずれのバッファも1個で示しているが，これが2個以上確保されている場合には移動方式の場合と同様に先読みあるいは後書き出しが行われ，CPU 処理と入出力オペレーションのオーバーラップがなされる．

前述のレジスター1の働きを念頭に再度図6-18を参照していただきたい．最初に実行される PUT は，単にレジスター1に出力バッファの最初のアドレスを入れているに過ぎない．次の LR 命令でレジスター1のアドレスをレジスター6に移している．次の GET 命令でレジスター1に入っているアドレスがこわされてしまうからだ． GET 命令で，バッファ内の処理すべき論理レコードのアドレスがレジスター1に入れられる． MVC 命令では， GET マクロ命令で得られた論理レコードのアドレス（0(1)）をもとにし，レジスター6に入っている出力バッファ内のアドレス（0(80,6)）へ80バイト移動させている．このようにバッファ内の論理レコードの位置がレジスター1に示される方式が位置指定方式の GET および PUT マクロ命令である．移動方式のような余計な作業域が不要であり，かつ作業域へデータを移動するために要する時間も必要ない．

3)　位置指定方式の入力とレコードの更新

直接アクセス・ボリュームにあるデータ・セットのレコードを入力して更新処理を行い，更新処理が行われたブロックだけをもとの場所に書きもどすときには，位置指定方式の GET マクロ命令と PUTX マクロ命令を組み合わせて使用する． GET マクロ命令で物理レコードをバッファ内に読み込み，論理レコードのアドレスがレジスター1に入れられる．このアドレスをもとに更新の対象となるレコードを選び出し更新作業を行ったうえで，更新レコードに対してのみ PUTX マクロ命令を与える．

レコードの更新の例を示したのが**図6-20**である．更新の対象になるデータ・セットに対しては，位置指定方式の GET と PUTX マクロ命令の両方を使用

```
          :
          OPEN    (INDCB,, UPDCB, (UPDAT))
          GET     UPDCB
     ┌─────────────────┐
     │ レコードの更新処理 │
     └─────────────────┘
          PUTX    UPDCB
          :
UPDCB     DCB     DSORG=PS, MACRF=(GL, PL), DDNAME=UPDD,        C
                  RECFM=FB, BLKSIZE=500, LRECL=100,             C
                  EODAD=LAST
          :
```

図 6-20 PUTX マクロ命令を使ったレコードの更新－位置指定方式

するので，DCB のパラメータで MACRF=(GL, PL) と指定する必要がある．また，OPEN マクロ命令のオプションでは，UPDAT と指定する．図では，更新処理の部分のコーディングは全く省略しているが，ここでは通常の更新処理で行うと全く同じような手順が入る．更新処理を行ったレコードに対してだけ PUTX マクロ命令が働くようにしておけばよい．

4) 置換方式による処理

置換方式のデータ転送は，交換バッファ割当ての手法で実現される．**置換方式** (substitute mode) は前述の位置指定方式に類似する点があるが，プログラマーがバッファ・セグメントに等しい作業域を用意しておく必要がある．この作業域がバッファ・セグメントに置き換わる．つまり，入出力バッファとあらかじめ用意した作業域とが必要に応じてその役割りを交替しながら，入出力オペレーションが遂行されてゆく．置換方式は，主記憶装置内で論理レコードの移動を伴わないで処理が行えるような方法である．つまり，入力論理レコードで占められているバッファ・セグメントがある時点で作業域になり，別の時点ではそのまま出力論理レコードのバッファ・セグメントに姿をかえるのである．

置換方式の例を示したのが図 6-21 である．処理の内容は図 6-11 で使ったも

```
                 :
         OPEN    (INDCB,, OUTDCB, (OUTPUT))
         LA      0, WORKAREA  作業域のアドレスをレジスター0に
                              入れる.
LOOP     GET     INDCB, (0)   置換方式
         LR      0, 1         作業域のアドレスをレジスター0に
                              入れる.
         PUT     OUTDCB, (0)  置換方式
         LR      0, 1         作業域のアドレスをレジスター0に
                              入れる.
         B       LOOP
ERROR    :
                  } 入出力エラーの処理
         :
         RETURN                制御権を制御プログラムに戻す.
LAST     CLOSE   (INDCB,, OUTDCB)
         :
         DS      0D
WORKAREA DS      CL80
INDCB    DCB     DSORG=PS, MACRF=(GT), DDNAME=INDD,     C
                 BFTEK=E, BFALN=D, SYNAD=ERROR,         C
                 RECFM=F, BLKSIZE=80, EODAD=LAST
OUTDCB   DCB     DSORG=PS, MACRF=(PT), DDNAME=OUTDD,    C
                 BFTEK=E, BFALN=D, EROPT=ACC,           C
                 RECFM=FB, BLKSIZE=240, LRECL=80

          :
```

図 6-21　順次データ・セットの処理－置換方式

のと全く同じである.

DCB マクロ命令のパラメータでは，**交換バッファ割り当て手法** (exchange buffering technique) を適用することを BFTEK=E で指示する．また置換方式の GET および PUT 命令を使うことを MACRF=(GT) ないしは MACRF=(PT) で要求する.

GET マクロ命令を与えるに先立って，プログラマーが設定した作業域のアドレスをレジスターにロードしておき，GET マクロ命令ではそのレジスターを指示する．指示された作業域のアドレスは，GET 命令が実行されることに

より，つぎの入力論理レコードのアドレスと役割りを交換し，入力論理レコードのアドレスがレジスター1に返される．このレジスター1のアドレスにもとづいて入力レコードの処理を行えばよいわけだ．入力レコードの処理作業が終ったあとで，このアドレスをPUTマクロ命令に渡せばよい．ただし，図6-21の例では入力レコードをそのまま出力するのであるから，何らの処理を行う必要もない．それゆえ，入力レコードが入っている作業域のアドレスをPUT命令に受け渡すだけでよい．そのために，レジスター1に入っているアドレスをレジスター0に入れたうえでPUTマクロ命令を実行している．PUT命令の2番目のオペランドで出力すべき理論レコードのアドレスを指示している．出力レコードのアドレスは，作業域として使用可能な出力バッファ・セグメントのそれと役割りを交替する．そして，次の作業域の役割りを果すセグメントのアドレスは，レジスター1にもどされる．したがって，このアドレスをGET命令に受け渡すためにレジスター0に入れたうえで，GET命令の実行に移る．

このような方法で，各バッファ・セグメントが入力域，作業域，出力域へと役割りを変えてゆくので，主記憶域内でのレコードの移動は全く伴はない．つまり，出力レコードはもとの入力レコードの記憶域からそのまま出力されるのである．しかし，論理的には何時でも正しいデータ・セットとの対応づけがなされているのである．なお，置換方式のバッファ・セグメントの役割りがどのように変ってゆくかの具体例は，第5章の「九．バッファの確保と制御機能」の「交換バッファの割当て」という項目を参照されたい．

図6-21のDCBマクロ命令には，これまでの例で示さなかった新しい2つのパラメータがある．SYNAD=ERRORとEROPT=ACCである．これについて若干説明しておこう．これら2つのパラメータはいずれも入出力エラーが生じ，システムで回復不可能と判定した場合の措置の仕方を指示するためのものである．SYNADパラメータでは，ユーザ・ルーチンのアドレスを指示しユーザ・ルーチンでエラーの分析と処理を行うことを要求する．EROPTパラメータは，システムに処理を依頼するためのものであり，ACCと指定するとエラーが生じたブロックをそのまま処理することを，SKPとするとエラーが生じたブロックは無視してしまうことを，またABEとするとそのタスクの実行を

停止することを要求する．

この例では，交換バッファ割当てと置換方式と関係づけたものであったが，交換バッファ割当ては単に置換方式だけに使われるだけではない．移動方式や位置指定方式でも利用できることをつけ加えておく．

5）制御文字を使った場合の処理

本章の第2節「データ・セットのレコード形式」の説明の中で，いずれのレコード形式においてもデータ・レコードの直前に1バイトの制御文字を挿入することができることについてふれた．この制御文字は，本来印刷装置への出力においてスペースとスキップを制御し，またカード穿孔において穿孔カードのスタッカー選択に使うためのものである．したがって，印刷装置やカード穿孔装置にレコードを直接出力する場合だと，制御文字はスペースやスキップあるいはスタッカー選択のコントロールに用いられるのであって，出力データの一部とはならない．しかし，印刷データや穿孔データを一たん磁気テープ・ボリュームや DASD ボリューム上に出力する方法をとる場合には，制御文字はレコードの1部としてデータ・レコードの直前の1バイトを占めることになる．

ところで，データ管理プログラムにおいて制御文字は QSAM だけが特有の扱いをうけるのではない． BSAM と BPAM においても QSAM の場合と全く同様の処理が行われる．説明の便宜上 QSAM の一部として取り扱っているに過ぎない．

制御文字を使用する場合には，DCB マクロ命令の RECFM パラメータの中でMまたはAを指定する．つまり，RECFM=FM, FA, FBM, FBA, VM, VA, VBM, VBA などといった指定の中から，その時々の処理に合致した形式を指示する．この場合，Mは制御文字として8ビットの機械コードを使用することを指示する．この機械コードは印刷装置や穿孔装置の種類ごとに決められており，指定の仕方は面倒である．そこで，もっと一般的指定の仕方として **ANSI** (American National Standard Institute) **コード**を使用する．制御文字としてこの ANSI コードを使用することを宣言するのが，Aである．ANSI コードの一覧表は**表 6-4** に示した通りである．表の中の b は制御文字としてブランク

制御文字	制 御 文 字 の 意 味
b	印刷する前に1スペースをとる
0	印刷する前に2スペースをとる
−	印刷する前に3スペースをとる
＋	印刷する前にスペースをとらない
1	チャネル1にとぶ
2	チャネル2にとぶ
3	チャネル3にとぶ
4	チャネル4にとぶ
5	チャネル5にとぶ
6	チャネル6にとぶ
7	チャネル7にとぶ
8	チャネル8にとぶ
9	チャネル9にとぶ
A	チャネル10にとぶ
B	チャネル11にとぶ
C	チャネル12にとぶ
V	穿孔したカードをスタッカーのポケット1におとす
W	穿孔したカードをスタッカーのポケット2におとす

表 6-4 ANSI コードの制御文字一覧表

を用いることを意味している．

制御文字としてANSIコードを使った場合のコーディング例を示したのが，**図6-22**である． DCB マクロ命令の中の RECFM＝FA という指定で，固定長非ブロック化レコードの形式でレコードの先頭に1バイトの ANSI コードを制御文字として挿入することを指示している．出力レコードを構成する場合には，その記憶域として制御文字用の1バイトを余分に設けると同時に，レコードを出力する前に，つまり PUT マクロ命令を与える前に該当個所に制御文字を挿入する必要がある．また，BLKSIZE および LRECL では制御文字1バイト分を加味した長さを使用することは言うまでもない．

印刷装置や穿孔装置に直接出力する場合には，制御文字はアクセス方式ルーチンによって **CCW** (Channel Command Word) に変えられ，出力装置のスキップ，スペースあるいはスタッカー選択の働きをする．したがって，出力レコードの1部とはならない．一方，磁気ディスク・ボリュームや磁気テープ・

```
              MVI     CTLCHAR, C '1'
              PUT     OUTDCB, OUTAREA
               .
               .
               .
              MVI     CTLCHAR, C '0'
              PUT     OUTDCB, OUTAREA
               .
               .
               .
OUTDCB        DCB     DSORG=PS, MACRF=(PM),                    C
                      RECFM=FA, BLKSIZE=133
               .
               .
OUTAREA       DS      0CL133
CTLCHAR       DS      C
DATA          DS      CL132
```

図 6-22　制御文字を使った場合の例

ボリュームに出力される時には，制御文字に何らの加工処理も行わずそのまま出力するので，レコードの1部として記録されてしまう．後刻，これらのデータ・セットを印刷装置や穿孔装置に出力する際に制御文字としての本来の機能を果すわけである．このことは，極めて重要な意味をもつ．ということは，制御文字を使うことによって，装置からの独立性が保たれるからである．印刷装置や穿孔装置に出力すべきデータ・セットであっても，同じプログラムを使って一時的に磁気テープや DASD 装置に出力しても何ら支障がないのである．

なお，装置からの独立性を要求しないプログラムにおいては制御文字を使う代りに CNTRL マクロ命令や PRTOV マクロ命令を使うことができる，CNTRL マクロ命令は，スタッカー選択，印刷装置のスペースとスキップ制御，さらには磁気テープのバックスペースや読みとばし（ブロック単位やファイル単位で）の機能を果す．一方，PRTOV マクロ命令には印刷出力だけに適用できるマクロ命令で，印刷装置のチャネル9とチャネル12をテストし，チャネル9や12が印刷位置にきている場合には，前述の CNTRL 命令を使って印刷用紙のスキップをする．

2. BSAM による順次データ・セットの処理

BSAM (Basic Sequential Access Method) では，READ および WRITE マクロ命令を使って，主記憶装置と補助記憶装置の間でのデータ転送を行う．しかし，これらのマクロ命令は入出力オペレーションの完了を同期化させる機能は持ち合わせていない．単に入出力オペレーションを開始させるに過ぎない．したがって，入出力オペレーションが正しく完了したかどうかは別のマクロ命令 CHECK ないし WAIT でテストする必要がある．QSAM の GET や PUT マクロ命令はレコードのブロッキングないしはディブロッキングの機能を果してくれたが，READ や WRITE 命令にはこのような機能はない．したがって，ユーザー・プログラムの中でブロックの組立てを行ったあとで WRITE マクロ命令を与える必要があるし，READ マクロ命令で読みとったブロックを論理レコードに分解して処理することもユーザー・プログラムに委ねられている．

BSAM には QSAM のような自動的な機能が少ない代りに，ユーザー・プログラムで意図したことを行うための自由度は高い．READ なり WRITE なりのマクロ命令と CHECK マクロ命令との間に別の処理ステップをを入れて

```
          :
          OPEN     (OUTDCB, (OUTPUT))
          :
CREATE    :        入力レコードを読み込み
          :          キーを KEYFLD に移し,
          :          DATAFLD に新しいレコードを作りあげる (ブロッ
          :          キングして).
          WRITE    OUTDECB, SF, OUTDCB, OUTBUF
          CHECK    OUTDECB
          B        CREATE
OUTBUF    DS       0D
KEYFLD    DS       CL8
DATAFLD   DS       CL200
OUTDCB    DCB      DSORG=PS, MACRF=(W), BLKSIZE=200,              C
                   KEYLEN=8, DDNAME=OUTDD, RECFM=F
          :
```

図 6-23 BSAM を使ったデータ・セットの作成 (1)

CPU 処理とのオーバラップをやらせたり，CHECK マクロ命令や WAIT マクロ命令を発するまえに複数個の READ ないしは WRITE マクロ命令を連続的に実行させて**連鎖スケジューリング**（chained scheduling）を行わせ，処理効率を高めることも可能だ（ただし，DCB マクロの NCP パラメータで2以上の数を指定する必要がある）．さらには，NOTE マクロ命令を使って読み書きしたブロックの相対アドレスを得たり，POINT マクロ命令で磁気テープや直接アクセス・ボリューム上のデータ・セット内の特定ブロックだけの読み書きをすることもできる．

なお，BSAM で用いる DCB マクロのパラメータの指定は基本的には QSAM の場合と変りない．ただし，BSAM では QSAM で説明した EROPT パラメータだけは使用できない．

1) BSAM によるデータ・セットの作成

BSAM を使ってデータ・セットを作成する場合の例を**図 6-23**で示している．この例では，直接アクセス・ボリューム上にカウント／キー／データ形式で出

〔記号名〕	READ WRITE	DECB 名，タイプ，DCB アドレス，バッファ・アドレス，{長さ / 'S'}

- DECB名…マクロ・エクスパンション（MACRO expansion）の1部として生成される DECB（Data Event Control Block）に割当てるべき記号名
- タイプ…READ/WRITE マクロ命令のタイプ．つぎのいづれかのコードを記入する．ただし，WRITE マクロ命令では SF だけ．
  - SF…通常の読み書き．
  - SB…磁気テープのデータ・セットを逆読みする場合に指定．
- DCB アドレス…読み書きするデータ制御ブロック（DCB）のアドレス．
- バッファ・アドレス…入出力ブロックを収容するための主記憶域のアドレス．直接アクセス・ボリュームでキーを使用する場合は，キー・フィールドのエリアの先頭に置くこと．
- 長さ…読み書きするブロックのデータ・バイト数．不定長レコード形式（RECFM＝U）のときだけ指定．
  'S' を指定すると，DCB の中の DCBBLKSIZE フィールドの値が使われる．

表 6-5　READ/WRITE マクロ命令の一般形式

力している．キーは8バイトの長さ（KEYLEN=8）である．なお，コーディングは出力に関係する部分だけにとどめてある．

DCB マクロ命令の MACRF パラメータでは WRITE マクロ命令を使うことを要求する．ブロッキングの機能はないので LRECL を指定する必要はない．WRITE マクロ命令の一般的な指定形式は，**表 6-5** に示した通りである．READ マクロ命令の場合にもほとんど同じ形式である．DECB 自体は，WRITE マクロ命令によって自動的に作り出されるのであるが，その DECB につける記号名は WRITE 命令のオペランドで与えてあとで参照できるようにする．使用するバッファは，必ず WRITE マクロ命令の4番目のオペランドで指定する必要がある．例で示しているように BSAM ではバッファ・プール以外にユーザー・プログラムで確保した記憶域をバッファとして指定することが許される．

WRITE マクロ命令には，ブロッキングの機能がないので，ユーザー・プログラムでバッファにブロックを組立てたうえで WRITE 命令を実行させるようにコーディングする．なお，WRITE オペレーションが正しく完了したかどうかは，CHECK マクロ命令でテストする．この命令が与えられることにより，出力オペレーションが終了するまでプロセスは待たされる．出力オペレーションが終了すると CHECK ルーチンで入出力エラーおよび例外事態の発生の有無がチェックされ，しかるべき手順がとられる．回復不可能なエラーがあると DCB マクロ命令の SYNAD パラメータで指示されたユーザー・ルーチンに制御権を渡す．この例では，SYNAD ルーチンが指定されていないのでエラーがあるとこのプログラムの実行を放棄してしまうことになる．なお，WRITE 命令と CHECK 命令の間に目下のオペレーションにかかわりをもたない別の処理ルーチンを入れることによって，CPU オペレーションと出力オペレーションのオーバラップが実現できる．

データ・セットを作成するさいに，どのブロックがどの位置に書き出されたかの記録をとっておくと，後日そのデータ・セットの特定レコードないしは特定ブロックだけを取り出すのに好都合である．この場合，各ブロック内のレコードの一番大きいキーとそのブロックが書き出された相対的な位置（アドレ

```
                 :
                OPEN     (OUTDCB, (OUTPUT))
                 :
                LA       5, TABLE
CREATE           :       新しいレコードの作りあげ
                WRITE    OUTDECB, SF, OUTDCB, OUTBUF
                CHECK    OUTDECB
                NOTE     OUTDCB      レコードの相対アドレスをレジスター1へ
                MVC      0 (8, 5), KEYFLD      テーブルにキーを入れる
                ST       1, 8 (5)            テーブルに相対アドレスを入れる
                LA       5, 12 (5)     レジスター5のアドレスを12バイト増やす
                B        CREATE
OUTBUF          DS       0D
KEYFLD          DS       CL8
DATAFLD         DS       CL200
TABLE           DS       150F    50個のテーブル・エントリーを確保
OUTDCB          DCB      DSORG=PS, MACRF=(WP), BLKSIZE=200,           C
                         KEYLEN=8, DDNAME=OUTDD, RECFM=F
                 :
```

図 6-24 BSAM を使ったデータ・セットの作成 (2)

ス) とを対比したテーブルにしておくのが普通である. この例を示したのが**図6-24**であり, 図6-23の問題にテーブル作成の手順を付加しただけである.

読み書きが行われたブロックの相対的位置を返してくれるマクロ命令が, NOTE である. READ/WRITE オペレーションの完了を確認したあとでこのマクロ命令を実行させると, 目下のブロックの相対的位置がレジスター1に返される. この命令は, 磁気テープおよび DASD 上のボリュームに対してのみ有効である. 磁気テープ・ボリュームの場合には, ブロック番号が2進数でレジスター1に返される. DASD の場合には, 相対的トラック・アドレスの形式 (TTR) でレジスター1に返される. 相対的トラック・アドレスに関しては, 第4章の「七. 直接アクセス・ボリュームのスペース割当て機能」の「1. DASD のハードウェア特性」を参照されたい.

図6-24では, テーブルのアドレスをレジスター5に挿入し, これを基点として, 各ブロックごとのキーと相対的トラック・アドレスを対にしてテーブルを

作成している. 各ブロックごとのテーブル・エントリーを作るごとにレジスター5のアドレスを12バイト増やすことによってアドレスを調整しているのである. なお, DCB マクロ命令の MACRF で WP と指定されていることに注意されたい. これは, WRITE と NOTE の両方のマクロ命令を利用することを要求しているのである.

2) BSAM を使ったレコードの直接取り出し

順次データ・セット内のブロックの相対的位置がわかっていれば, それにもとづいて特定ブロックだけを取り出して処理することが可能になる. このために使用できるのが, POINT マクロ命令である.

POINT マクロ命令を使用すると, データ・セット内の特定ブロックから次

```
                 :
         OPEN    (INDCB)
                 :
         LA      5,0     インデックス・レジスターを0に初期値化
NGROUP   POINT   INDCB, TABLE+8(5)
SGROUP   READ    MYDECB, SF, INDCB, INBUF
         CHECK   MYDECB
                 :       レコードの処理
         BE      NEXT    グループ内の最後のレコードまで処理したとき
         B       SGROUP      グループ内の未処理レコードがあるとき
NEXT     LA      5, 12(5)  インデックス・レジスターを12バイト増やす.
         B       NGROUP
LAST     CLOSE   (INDCB)
                 :
INBUF    DS      0D
KEY      DS      CL8
DATAFLD  DS      CL200
TABLE    DS      150F
INDCB    DCB     DSORG=PS, MACRF=(RP), BLKSIZE=200,          C
                 KEYLEN=8, DDNAME=INDD, RECFM=F,             C
                 EODAD=LAST
                 :
```

図 6-25 BSAM を使ったレコードの直接取り出し

の READ/WRITE オペレーションを開始させるよう，制御プログラムに要求できる．この例を示したのが，**図6-25**である．POINT マクロ命令のオペランドでは DCB のアドレスと特定ブロックの相対的アドレスの入っているフルワードの記憶域とを指示する．図6-25では相対的アドレスとして図6-24で作りあげたテーブルをそのまま使っている．したがって，各テーブル・エントリーが8バイトのキーとそれにひきつづくフルワードの相対的トラック・アドレスを想定してコーディングしている．しかも，テーブルはあらかじめ主記憶装置に読み込まれているものと仮定している．

例では，テーブル内で示されている相対アドレスに対応する全ブロックの論理レコードを処理しようとしている．特定のブロックだけを選んで処理する方法であっても一向に差しつかえないし，特定ブロックから開始してある範囲のブロックをすべて処理する方法だってとれる．特定ブロックが見つかるまでユーザー・プログラムで検索してゆくやり方よりは，はるかに効率的である．

なお，DCB マクロ命令の MACRF パラメータで RP としているのは，READ と POINT マクロ命令を必要とする旨を指示しているのである．

3) BSAM を使ったレコードの更新

QSAM においては，PUTX マクロ命令を位置指定方式で使って更新レコードを直接アクセス・ボリューム上の元の位置に書きもどすことができた．BSAM においては同じことを WRITE マクロ命令を使って行う．READ したと同じデータ・セットに対して WRITE マクロ命令を与えることによって，データ・セットの元の場所にレコードを書きもどす．

この例が**図6-26**である．更新用データ・セットを OPEN するには，QSAM の場合と同じように OPEN マクロ命令のオペランドで UPDAT オプションを指示する．また，DCB マクロ命令の MACRF パラメータでは同一データ・セットに対して READ と WRITE マクロ命令を使ってレコードの更新を行うことを (R, W) で指示し，更新用のアクセス方式ルーチンを要求する．

データ・セットから READ マクロ命令でブロックを読み込んだあと，ディブロッキングを行いながら特定論理レコードの更新作業を行う．そのあとで，

```
              :
           OPEN     (INDCB, (UPDAT))
              :
NEXT       READ     MYDECB, SF, INDCB, BUFFER
           CHECK    MYDECB
              ↓     バッファ内でレコードの更新作業
           WRITE    UPDECB, SF, INDCB, BUFFER
           CHECK    UPDECB
           B        NEXT
ERROR         ↓     入出力エラー処理
LAST       CLOSE    (INDCB)
              :
BUFFER     DS       0D
KEY        DS       CL8
DATAFLD    DS       CL200
INDCB      DCB      DSORG=PS, MACRF=(R, W), KEYLEN=8,              C
                    BLKSIZE=200, SYNAD=ERROR, RECFM=F,             C
                    DDNAME=UPDD, EODAD=LAST
              :
```

図 6-26　BSAM を使ったレコードの更新

更新レコードを含んでいるブロックに対してのみ WRITE マクロ命令を実行させる．このようにして，更新レコードをデータ・セットの元の場所に書きもどすことができるのである．なお，図 6-25 で示した POINT マクロ命令を使って特定ブロックだけを取り出して，更新作業を行うことも可能である．

五. 区分データ・セットの構成とその処理

1. 区分データ・セットの構成

区分データ・セットは，複数個の順次編成の**メンバー** (member) とそのメンバー名の登録簿とで構成される(図 6-27参照)．各メンバーは1個以上のレコードから成り立ち，それぞれメンバーごとに固有の名前が与えられており，それが**登録簿** (directory) に貯えられている．メンバー名で特定メンバーを選択することができ，メンバーを構成するレコードは順次的に記憶したり，取り出

されたりする.

区分データ・セットを利用する場合の主要な利点は，そのデータ・セットを一たんOPENすると，どのメンバーでも自由に取り出すことができることにある．それゆえに，プログラム・ライブラリーはすべて区分データ・セットで編成される．個々のプログラムやサブルーチンがメンバーであり，プログラム名やサブルーチン名がメンバー名として登録簿に貯えられる．このメンバー名を使って，必要とするプログラムやサブルーチンが自由に呼び込めるのである．また，個々のメンバーは必要のつど追加や削除を行うことができる．メンバーが削除されると，メンバー名だけが登録簿から除去され，そのメンバーで占められていたスペースはそのまま残ってしまう．つまり，データ・セットが再編成

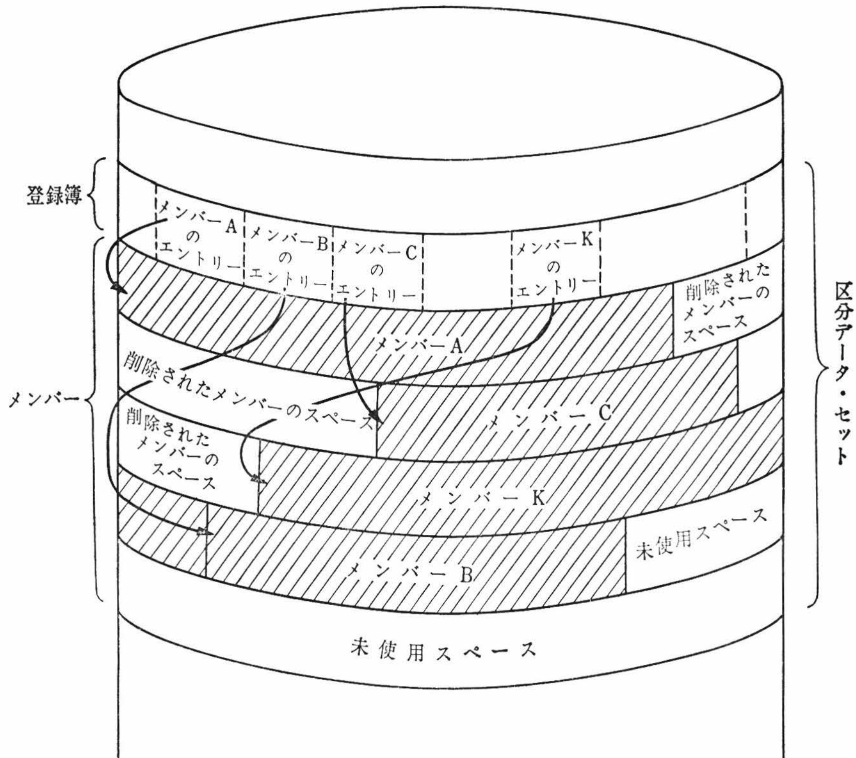


図 6-27　区分データ・セットの構成

し直されるまで再使用不可能である．ともあれ，区分データ・セットはプログラム・ライブラリーや SYS1. PROCLIB ないしは SYS1. PARMLIB のようなシステム・データ・セットに使われている．もちろん，ユーザー・デーク・セットの作成にも区分データ・セットは適用できる．各種のテーブルを集めて区分データ・セットにしておけば，種々の処理プログラムで必要なテーブルを参照するのに便利である．ユーザー・データ・セットとしての区分データ・セットの適用分野は他の編成のデータ・セットほど広くはないが，その構造や処理の仕方を知っておれば，制御プログラムがシステム・データ・セットやプログラム・ライブラリーを管理する仕組みを理解するのに大いに役立つ．

登録簿 (directory) は，図6-27で示しているようにデータ・セットの先頭にある一連のレコードで，各メンバーの**エントリー** (entry) を収容する．登録簿の各々のエントリーは，**メンバー名** (member name) とデータ・セット内でのメンバーの位置を示す情報で構成されている．さらに，62文字までのユーザー情報をつけ加えることもできる．なお，エントリーはメンバー名のアルファベット順に配列されている．

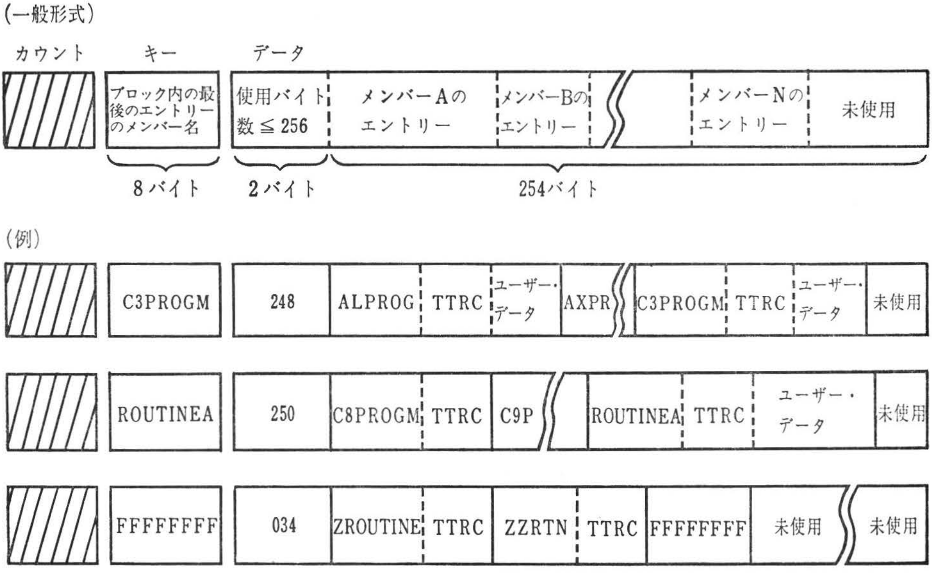


図 6-28 区分データ・セットの登録簿ブロックの形式

メンバーの位置を示すトラック・アドレスはシステムがエントリーに記録してくれる．このトラック・アドレスには絶対的トラック・アドレスではなく，データー・セット内での相対的位置を示すアドレスを使う．それゆえ，トラック・アドレスに何らの変更を加えることなく，データ・セット全体を移動しても支障が生じないのである．

区分データ・セットの登録簿が，直接アクセス・ボリューム上のデータ・セットに割り当てられた記憶域の先頭を占めることは前述した通りである．この登録簿は後述するような FIND と STOW マクロ命令によって検索されメインティナンスされる．登録簿の中のレコードは 256バイト単位の物理レコードに分かれており，各ブロックに複数個のメンバー・エントリーが収容されている（**図 6-28**参照）．各ブロックは，最大254バイトにおさまるだけのエントリーを収容し，残りのバイトは未使用のまま残される．ブロック内の実際に使用しているバイト数は，各ブロックの先頭の2バイトで示される．また，各ブロックの前にはキー・フィールドが設けられ，そのブロックに収められている最後のエントリーのメンバー名が貯えられる．したがって，メンバー名を使ってキー・サーチを行うことによって，当該メンバーのエントリーが収容されているブロックが容易につかめるのである．

各メンバー・エントリーには，メンバー名ないしは**別名**（alias）が収容されるわけであるが，各メンバーに対して最大16個の別名を与えることが可能である．別名は，それぞれ別個のエントリーを用いて与えるので，別名では異った相対トラック・アドレスを持たせることも可能となる．各メンバー・エントリーには，**図 6-29**に示したように，メンバー名（または別名）のほかにメンバーの相対トラック・アドレス，カウントおよびユーザー・データのフィールドがある．なお，図6-28の例で示しておいたように登録簿の最後のブロックの最後のエントリーのメンバー名のフィールドは全て1ビットであり，これによって，エントリーの終りを示している．

メンバー・エントリーは，最小12バイトであるが，ユーザー・データを付加することによって最大74バイトまで拡大することができる．ユーザー・データ域にはメンバーの属性などを示す情報を自由に入れることができる．ただし，

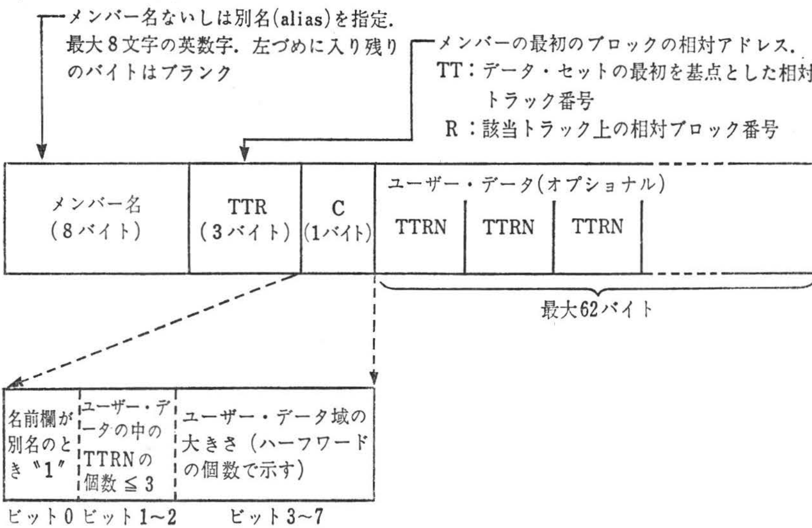


図 6-29　登録簿のメンバー・エントリーの形式

メンバー内の特定ブロックを指示するポインターを与える場合には，各ポインターは必ず4バイトにし，かつユーザー・データ・フィールドの先頭につめて入れる必要がある（図6-29参照）．ユーザー・データ・フィールドのポインターは，相対アドレス（TTR）であり，メンバー内の特定ブロックか**ノート・リスト**（note list）をさす（図6-30参照）．ノート・リストをポイントする場合には，4番目のバイトでノート・リストに含まれるポインターの個数を表わす．メンバー・ブロックを直接さす場合には0である．ノート・リストは図6-30で示すように同一メンバー内での追加のポインターを収容するものである．

このようにユーザー・データ・フィールド内のポインターやノート・リスト内のポインターを設けることによって，1つのメンバーの多様な参照の仕方が実現できるのである．なお，メンバー・エントリーはSTOW マクロ命令を使って登録簿に記入するが，ノート・リストをメンバー内に記録するのには通常のWRITE マクロ命令を使用する．また，相対アドレスを得るのにはブロックを書き出したあと NOTE マクロ命令を実行させることによって，レジスター1にそのアドレスを求める．これは，BSAM の個所で説明したマクロ命令であ

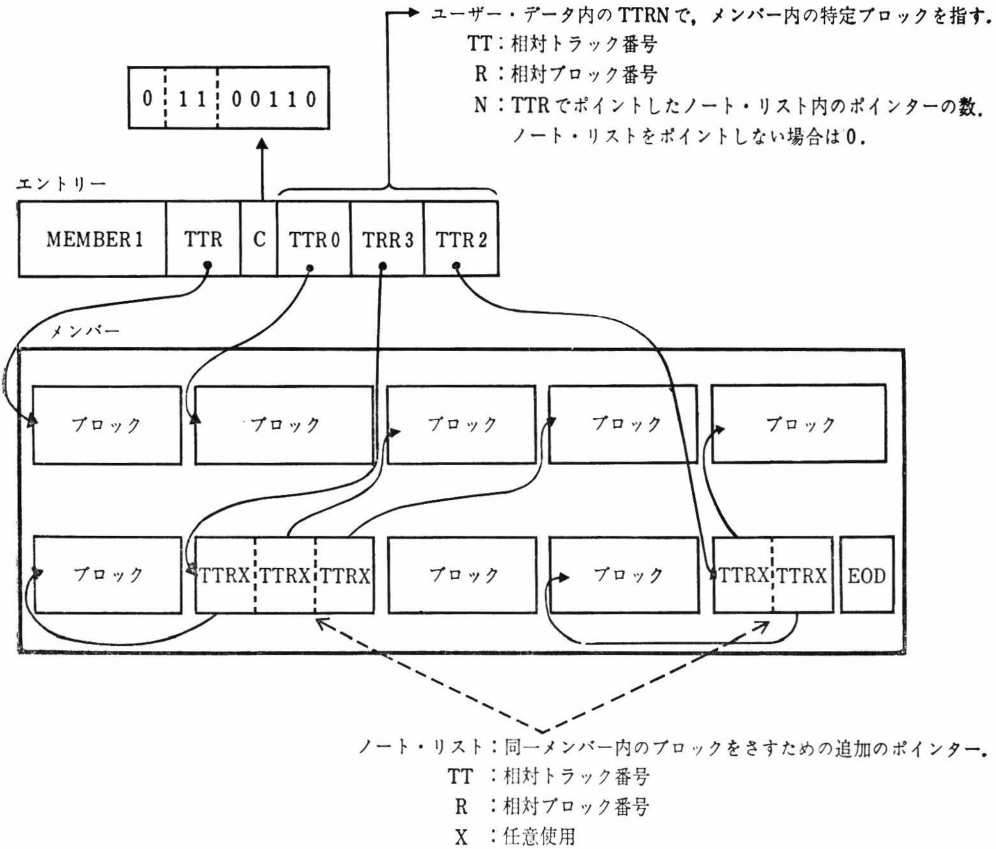


図 6-30　ユーザー・データ内の TTRN とノート・リストの相互関係

る．相対アドレスを得てメンバー・エントリーやノート・リストの各フィールドを作りあげる仕事は，プログラマーの責任である．

2.　区分データ・セットの処理

区分データー・セットの各メンバーは，順次編成になっている．したがって順次編成のデータ・セットと同じ様な方法で処理できる．基本アクセス技法でも待機アクセス技法でも処理できる．しかし，待機アクセス技法では登録簿の変更処理などはできない．

メンバーを探し出したり，登録簿を処理するために，いくつかの特別なマクロ命令が用意されている．BLDL マクロ命令は，主記憶装置内に登録簿のエントリー・リストを作りあげるのに使う．FIND マクロ命令は，特定メンバーを探し出すのに使う．また，STOW マクロ命令は，登録簿にメンバー名を追加したり，削除したり，変更したりするのに使う．それぞれのマクロ命令の具体的用法については後述するが，これらのマクロ命令を使うためには，DCB マクロ命令の DSORG パラメータで PO ないしは POU の指定が必要である．PO は Partitioned Organization，POU は Partitioned Organization Unmovable の意味である．Unmovable とは，データ・セットが装置の物理的位置に関する情報，つまり絶対アドレスを持っているため，そのままではデータ・セットを移動することができない性質である．なお，BLDL，FIND，STOW マクロ命令を与えるためには，すべての先行する入出力オペレーションが完了したことをチェックすることが必要である．

1） 区分データ・セットの作成

区分データ・セットを作成するのには，2つの方法がある．その1つは，QSAM または BSAM を使う方法であり，別のアプローチは BPAM を使う方法である．

登録簿にメンバー・エントリーを追加する必要がない場合には，図6-31に示すような方法で新しいデータ・セットを作りあず，最初のメンバーだけを書き出す．この場合には QSAM の PUT マクロ命令か BSAM の WRITE マクロ命令のいずれかでメンバーを書き出す方法をとる．処理の仕方の概略は次の通りである．

ジョブ制御カードの DD ステートメントで，新しい区分データ・セットのメンバーとしてデータを記憶する旨を示す．つまり，DNSAME＝ データ・セット名（メンバー名），DISP＝(NEW, KEEP) とする．同時にそのデータ・セットの登録簿とメンバーのために必要なスペースを SPACE＝(TRK, (100, 5, 7)) で要求する．最初に 100 トラック要求し，不足したら5トラックずつ増やす．登録簿としては，7ブロック分要求している．

DD ステートメント

```
//PSDD    DD     …, DSNAME=MASTFILE(INVENT), SPACE=        C
//               (TRK(100, 5, 7)), DISP=(NEW, KEEP)
```

処理プログラム

```
              :
         OPEN     (OUTDCB, (OUTPUT))
              :
              :
         PUT      OUTDCB    または WRITE マクロ命令
              :
         CLOSE    (OUTDCB)
              :
OUTDCB   DCB      DSORG=PS, DDNAME=PSDD, MACRF=(PL), ……
              :
```

図 6-31 区分データ・セットの作成 (1)

DCB マクロ命令では，DSORG パラメータで PS または PSU を指定する．一連の PUT または WRITE マクロ命令を使って，メンバーを構成するのに必要な全ブロックを書き出したあとで，CLOSE マクロ命令を与える．この場合，CLOSE マクロ命令を実行することによって自動的に STOW マクロのルーチンがとられ，登録簿にメンバー・エントリーが書き出される．

これらのステップを実行した結果，データ・セットとその登録簿が作成され，メンバーを構成するレコードが書き出され，エントリーが登録簿に作りあげられるのである．

この例では，1個のメンバーしか登録できない．したがって，この区分データ・セットにメンバーを追加するには，同じような手順をとらせる必要がある．ただし，追加の場合には各メンバーごとに別の DD ステートメントを用意し，DSNAME パラメータでそれぞれ固有のメンバー名を指定する必要がある．また，SPACE パラメータは省略し，DISP=(MOD, KEEP) とする必要がある．

QSAM や BSAM を使って区分データ・セットを作成する場合には，メンバー・エントリーが CLOSE マクロ命令で自動的に作成されてしまうので，メ

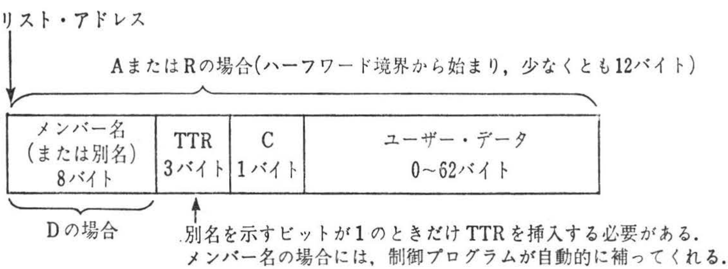
ンバー・エントリーのユーザー・データ・フィールドに情報を貯えるとか，ノート・リストを作成してポイントさせるようなことはできない．また，1つのプログラムで一度に複数個のメンバーを登録することもできない．このようなことを実現するためには，BPAM と STOW マクロ命令を使う方法をとる．

区分データ・セットにメンバーを一度に1つずつ追加する方法を採らない場合には，STOW マクロ命令を使ってメンバー・エントリーを登録簿に追加しなければならない．単に1個のメンバーを追加する場合だと，さきにみたよう

〔記号名〕	STOW	DCBアドレス，リスト・アドレス〔，登録簿アクション〕

●DCBアドレス……区分データ・セットに対応するDCBのアドレス
●リスト・アドレス…制御プログラムが登録簿をメインティナンスするのに必要な情報を貯えている主記憶域のアドレス．その形式や大きさは下記に示すように，指定する登録簿アクションのタイプに応じて異る．
●登録簿アクション…登録簿に対して行う処理のタイプをつぎのコードを使って指定する：
A－登録簿にエントリーを追加する．
C－登録簿内のメンバー名(または別名)を変更する．
D－登録簿内のエントリーを削除する．
R－登録簿内のエントリーの中味を置き換える．

〔リスト・アドレスの形式とサイズ〕



Cの場合

元のメンバー名 (または別名) 8バイト	新しいメンバー名 (または別名) 8バイト

図 6-32　STOW マクロ命令の一般形式

にデータ・セットを CLOSE するさいに STOW の機能が自動的に働き，メンバー・エントリーが登録簿に記入される．一方，STOW マクロ命令も１回に１個のメンバー・エントリーしか登録できないが，CLOSE マクロ命令と異なり，１つのプログラムの中で何回もくりかえし実行できるので，複数個のメンバー・エントリーが追加できる．STOW マクロ命令を使うとメンバー・エントリーにユーザー・データをつけ加えることができるし，登録簿内のメンバー名を削除したり，置き換えたりあるいは変更することもできる．またメンバー・エントリーを登録するのと全く同じ方法で別名を登録することもできる．なお，STOW マクロ命令の一般形式は**図 6-32**に示す通りである．

BPAM と STOW マクロ命令を使って区分データ・セットのメンバーを書き出す例を示したのが**図 6-33**である．その手順は次の通りである．

DD ステートメントでは，QSAM や BSAM を使うときのように DDNAME パラメータでメンバー名を指定しない．メンバー名は STOW マクロ命令のオペランドの１つであるリスト・アドレスで指示する．区分データ・セットのメンバーや登録簿に要するスペースはやはり DD ステートメントで要求する．

DCB マクロ命令では DSORG＝PO または DSORG＝POU の指定が必要である．メンバーを構成するレコードを書き出すためには，WRITE と CHECK マクロ命令を与える．これらの命令は，BSAM の場合と全く同じである．必要なら，NOTE マクロ命令を与えて，書き出したレコード・ブロックの相対アドレスを得る．このアドレスは，メンバー・エントリーのユーザー・データあるいはノート・リスト作成の情報として使う．なお，ノート・リストを作成する場合には，WRITE マクロ命令を使用する．ノート・リストもメンバーを構成するレコードであるからだ．なお，図 6-33ではノート・リストは作成していない．

全てのメンバー・レコードを書き出したあと，登録簿にメンバー名，相対アドレス，その他の情報を記入するために STOW マクロ命令を与える．ただし STOW マクロ命令で指定するリストには，あらかじめメンバー名，カウント情報あるいは必要ならユーザー・データを貯えておかなければならない．

このようにして１つのメンバーに対する登録が終ったら，他のメンバーのメ

DD ステートメント

```
// PDSDD     DD    …, DSNAME=MASTFILE, SPACE=(TRK, (100, 5, 7)), C
//                 DISP=MOD
```

処理プログラム

```
                 :
            OPEN      (OUTDCB, (OUTPUT))
                 :
            WRITE     OUTDECB, SF, OUTDCB, OUTBUF  メンバーの最初
                                                    のブロックの書き出し
            CHECH     OUTDECB
                 :
            WRITE     OUTDECB, SF, OUTDCB, OUTBUF  メンバーの2番
                                                   目のブロックの書き出し
            CHECH     OUTDECB
            NOTE      OUTDCB    ブロックの相対アドリスをレジスター1へ
            ST        1, DIRENTRY+12     相対アドレスをメンバー・エン
                                         トリーのユーザ・データ部分に
                 :                       入れる.
            MVC       DIRENTRY(8), =C'MEMBER10'
            MVC       DIRENTRY+11(1), =B'00100010'
            STOW      OUTDCB, DIRENTRY, A
                 :
                 :    } 他のメンバーの登録のために同様のことを行う.
                 :
            CLOSE     (OUTDCB)
                 :
            DS        0D
OUTBUF      DS        200F
DIRENTRY    DS        CL16
OUTDCB      DCB       DSORG=PO, DDNAME=PDSDD, MACRF=(W), …
                 :
```

図 6-33　区分データ・セットの作成 (2)

ンバー・レコードとエントリーを書き出してゆくために再び WRITE, CHECK, NOTE あるいは STOW マクロ命令を与えてゆく. この方法を用いると1つのプログラムで複数個のメンバーが書き出せるし, メンバー・エントリーの追加情報を自由に与えることができる. また, ノート・リストを作成し, そ

のポインターをメンバー・エントリーに入れることもできるのである.

2)　区分データ・セットのメンバーの取出し

メンバーの取出しにも，区分データ・セットの作成の場合と同様2つのアプローチが考えられる. QSAM ないしは BSAM を使ってメンバーを取り出す方法と BPAM を使ってメンバーを取り出す方法である.

QSAM ないしは BSAM を使う場合には，1つのプログラムで1回に1個の特定メンバーだけしか取り出せない. しかも，メンバー・エントリーのユーザー・データの情報を駆使した処理を行うとか，ノート・リストの情報を使った特定ブロックの取り出しは出来ない. したがって，BPAM を使う場合のような自由度はプログラマーに与えられない. ともあれ，この方法でのメンバーの取り出しの例をみよう. **図 6-34**がそれである.

DD ステートメントの DSNAME パラメータではデータ・セット名だけでなく，かつこの中で取り出したい特定のメンバー名も指定する. 出来合いのデータ・セットからメンバーを取り出すのであるから，DISP=OLD である. DCB マクロ命令は DSORG=PS ないし PSU とする. OPEN マクロ命令を与え，一連の GET ないし READ マクロ命令を与えることによってメンバーを構成するすべてのレコード・ブロックを取り出して処理する. 処理が終った

DD ステートメント

```
// PDSDD  DD     …, DSNAME=MASTFILE(MEMBERK),  DISP=OLD
```

処理プログラム

```
              :
              OPEN     (INDCB)
              GET      INDCB          または，READ マクロ命令
               ↓       処理
              CLOSE    (INDCB)
              :
     INDCB    DCB      DSORG=PS,  DDNAME=PDSDD,  MACRF=(GL),
              :
```

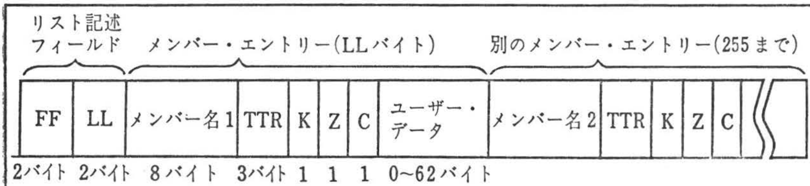
図 6-34　区分データ・セットのメンバー取出し (1)

ら CLOSE マクロ命令を与えればよい.

プログラムが実行されると，自動的に登録簿が検索される. DD ステートメントの DSNAME で指定されたメンバー名にもとづいて検索し，そのメンバーの位置を示す情報が DCB に収容される. この情報，つまり相対アドレスを起点にレコード・ブロックの読み込みが行われるのである.

CLOSE したり，再度 OPEN したりしないでいくつかのメンバーを処理したり，登録簿内の追加のデータを利用したりするためには BPAM を使う. この場合，必要なメンバーのエントリー情報を得るために BLDL マクロ命令を使う. また，特定メンバーの検索を行うために FIND マクロ命令を用いる.

BLDL マクロ命令は，区分データ・セットの登録簿内のエントリーの情報を主記憶装置に設けたリスト（図 6-35参照）に入れてくれる. BLDL マクロ命令のオペランドでは，区分データ・セットの DCB アドレスと主記憶域にプログラマーが設けたリストのアドレスとを指定する. リストの形式は図6-35に示しているように登録簿のそれによく似ている. リスト記述フィールドの情報とメンバー名は BLDL マクロ命令を実行する前にプログラマーが与える. BLDL マクロ命令が実行されると，リスト内の各メンバー名に対応する登録簿エント



BLDLマクロ命令を実行する前にプログラマーが与える必要のある情報:
- FF＝リスト内のメンバー・エントリーの数
- LL＝各メンバー・エントリーのバイト数≧12バイト
- メンバー名＝8バイト，左づめ. 別名であってもよい

BLDLマクロ命令が実行されることによって得られる情報:
- TTR＝メンバーの開始位置，つまり相対アドレス
- K ＝データ・セットだけのとき0
- Z ＝バウンダリー調整のため0が入る
- C ＝登録簿のメンバー・エントリーのCフィールドと同じ内容
- ユーザー・データ＝エントリーに入るだけのユーザー・データ

図 6-35 BLDL マクロ命令で使用するリストの形式

リーのメンバー・アドレスとユーザー・データとがリストの該当個所に挿入される．なお，リスト内に2個以上のメンバー名を入れる場合には，アルファベット順にしておく必要がある．ともあれ，BLDL マクロ命令を実行することによって登録簿のエントリー情報がリスト内に貯えられるので，このリストに対して FIND マクロ命令を出すことによって，メンバーの取り出し時間を短縮することができるのである．

FIND マクロ命令は，指定された区分データ・セットのメンバーの最初のブロック・アドレスを決定するのに使う．このアドレスが，つぎの READ マクロ命令の読み取りの開始アドレスに適用される．1個のメンバーだけを探し出したいのであれば，前述したように DD ステートメントでデータ・セット名とメンバー名を指定すれば，この FIND 機能は自動的に達成される．つまりシステムが指定されたメンバーのブロック・アドレスを DCB に入れてくれるので，後続の GET ないしは READ マクロ命令で所定の位置から処理が始められた．しかし，1つのプログラムの中で2個以上のメンバーを探し出して処理するのには FIND マクロ命令を使わざるをえない．このマクロ命令の形式を示したのが，**図 6-36**である．図 6-36のマクロ命令で分るように，制御プログラムがメンバーを探し当てる方法には2通りある．その1つは，メンバー名を収容した記憶域のアドレスを指定する方法である．この場合には該当区分データ

〔記号名〕	FIND	DCB アドレス，{ メンバー名アドレス，D / 相対アドレス・リスト，C }

◦DCB アドレス……処理すべき区分データ・セットの DCB アドレス

◦メンバー名アドレス，D……メンバー名が収容されたダブルワードのアドレス．メンバー名（または別名）は左づめで入れておくこと．制御プログラムは，このメンバー名にもとづいて指定されたデータ・セットの登録簿をサーチし，相対アドレスを得る．

◦相対アドレス・リスト，C……主記憶装置にある BLDL リストの相対アドレス部分（TTRK）のアドレス．制御プログラムは，与えられた相対アドレスをもとに直ちに次の READ オペレーションを可能にする．

図 6-36　FIND マクロ命令の一般形式

タ・セットの登録簿を最初からサーチし，メンバー名に対応する相対アドレスを得る．それだけに余計の時間を要する．他の方法は，BLDL マクロ命令で得た相対アドレスを用いるのでサーチ時間は全く必要としない．したがって，一段と効率的になる．

さて，BLDL マクロ命令と FIND 命令を使ってメンバーを取り出す方法をみよう．図6-37がこの例である． DD ステートメントの DSNAME ではデータ・セット名だけを指定する． DISP パラメータでは OLD を指定する．またDCB マクロ命令で DSORG=PO ないしは POU を指示する．登録簿から所要メンバー・エントリーの情報を得るために BLDL マクロ命令を与える．メンバー名 MEMBERK のエントリー情報だけを得ている．そのあとで，メンバー・レコードを読み取る場合の準備をするために FIND マクロ命令を与

DD ステートメント

```
//PDSDD     DD    ……, DSNAME=MASTFILE, DISP=OLD
```

処理プログラム

```
                    :
                 OPEN     (INDCB)
                    ↓
                 BLDL     INDCB, LISTADDR
                 FIND     INDCB, LISTADDR+12,C
                 READ     INDECB, SF, INDCB, INBUF
                 CHECK    INDECB
                    ↓     処理
                 CLOSE    (INDCB)
                    :
LISTADDR         DS       0H
                 DC       H'1'
                 DC       H'12'
                 DS       CL12'MEMBERK'
INBUF            DS       200F
INDCB            DCB      DSORG=PO, DDNAME=PDSDD, MACRF=(R), ……
                    :
```

図 6-37 区分データ・セットのメンバーの出取し (2)

える．FIND の2番目のオペランドでは BLDL マクロ命令で得られた相対アドレスの入っている場所を指定する．なお，この場合には FIND マクロ命令の代りに POINT マクロ命令を使っても同じ結果になる．

次の READ マクロ命令ではメンバーの最初のレコード・ブロックが取り出される．一連の READ および CHECK マクロ命令を発して，メンバーを構成するすべてのレコード・ブロックを取り出し，処理してゆけばよい．このようなステップを取り出すすべてのメンバーに対してくり返せば，1つのプログラムの中で必要な数だけのメンバーの取り出しと処理が実現できるのである．

なお，BLDL マクロ命令で得られたメンバー・エントリーのユーザー・データのうちの相対アドレスを使って POINT すれば，別のレコード・ブロックから取り出しを開始したり，ノート・リストだけを取り出すこともできる．ノート・リストの中の相対アドレス使って再び POINT すれば，次の READ マクロ命令ではさらに別個のメンバー・ブロックから読み出しが可能となる．このように BPAM を使うと，目的に応じて多様なメンバー・レコードの取り出しが可能になる．

これまで，区分データ・セットの作成とメンバーの取り出しを中心に区分データ・セットの処理の仕方をみてきた．これらの処理以外に区分データ・セットのメンバーを更新したり，特定メンバーを削除したり，新しいメンバーで置き換えたりすることもできる．なお，区分データ・セットの取り出しや処理に使われるいくつかのユーティリティ・プログラムの機能があるので簡単にまとめておこう．

- IEHMOVE……区分データ・セットの再編成に使われる．データ・セットは別の DASD 記憶域に移され，メンバー自体もメンバー名のアルファベット順に並べかえられる．メンバーが削除されたことによって生じていた空きのスペースは自動的につめられる．
- IEBCOPY……区分データ・セットの再編成に使われるが，IEHMOVE と異なりその場で再編成が行われる．しかも，メンバーの並べ換えは行われない．単に削除メンバーのスペースが自動的につめられるだけである．したがって，メンバー自身が使用頻度順に

配列されている場合の再編成に向いている．

●IEBUPDTE……区分データ・セットの更新や，メンバーに対して**別名**(alias) を与えるのに使われる．

六．索引順次データ・セットの構成とその処理

索引順次データ・セット (indexed sequential data set) を用いると，非常に弾力的な処理オペレーションができる．索引順次データ・セットは順次編成データ・セットの特徴と直接アクセス・データ・セットの特徴とを兼ね備えている．したがって，データ・セットを先頭から順番に読み書きすることができるし，特定レコードを直接アクセスして処理することもできる．新しいレコードを追加したり，不要のレコードを削除することも容易に行える．システムが自動的にレコードを追加する個所を探しだしたり，レコードを削除するさいに必要となる調整をしてくれるからである．

索引順次データを処理するためのアクセス方式としては，**QISAM** (Queued Index Sequential Access Method) と BISAM (Basic Index Sequential Access Method) とが用意されている．しかも，それぞれのアクセス方式には別々の異った機能がもたらされている．区分データ・セットのように異ったアクセス方式が同一目的の処理に使えるわけではない．つまり，索引順次データ・セットを作成するためには QISAM しか使えない．また，データ・セットを順次的に処理したり，更新したりする仕事にも QISAM を使う．さらに，データ・セットの終りに新規レコードを追加するのにも QISAM が使われる．一方，BISAM はデータ・セット内のレコードをランダムに取り出したり，更新を行うのに用いる．また，既存のレコードの途中に新しいレコードを挿入するのにも BISAM が機能する．

1. 索引順次データ・セットの構成

索引順次データ・セットは，その名前が示す通り各レベルの**索引** (index) とキーの順番に並べられたレコード群とから構成されている．つまり，索引付き

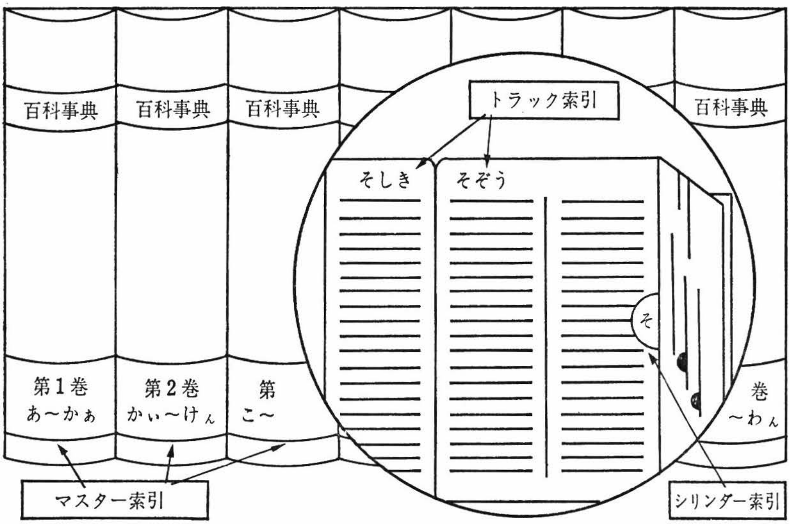


図 6-38　索引順次データ・セット編成の概念

の順次編成データ・セットである．身近な例で考えると，百科事典の構成に近い（**図 6-38**参照）．

そもそも百科事典は，順番に読んでいくことを想定して編さんされたものではない．その時々に生じた問題をランダムに調べてゆくことを目的にしたものである．しかし，検索を用意にし，かつ新規用語の追加や不要な用語の削除を正確・迅速に行うことをねらいとして，各用語はアイウエア順に配列されている．そのうえで，いくつかのレベルの索引を用意して，用語を探し出しやすくしている．このような百科事典とほとんど同じ構想を導入したのが索引順次ファイルである．しかし，直接アクセス・ボリュームをデータの記憶媒体にするために，その物理的性格からデータ・セットの構造は百科事典のそれとは大幅に異ならざるを得ない．基本的考え方においてはかなり共通点が見出せるのである．

索引順次データ・セットの基本構造を示すと**図 6-39**のようになる．**索引域**（index area），**主要データ域**（prime data area）および**あふれ域**（overflow area）の3つの記憶域から構成される．主要データ域にはデータ・レコードが記憶される．索引域は，トラック索引，シリンダー索引およびマスター索引のそれぞれの区域からなり立っており，データ・レコードの検索を容易にするた

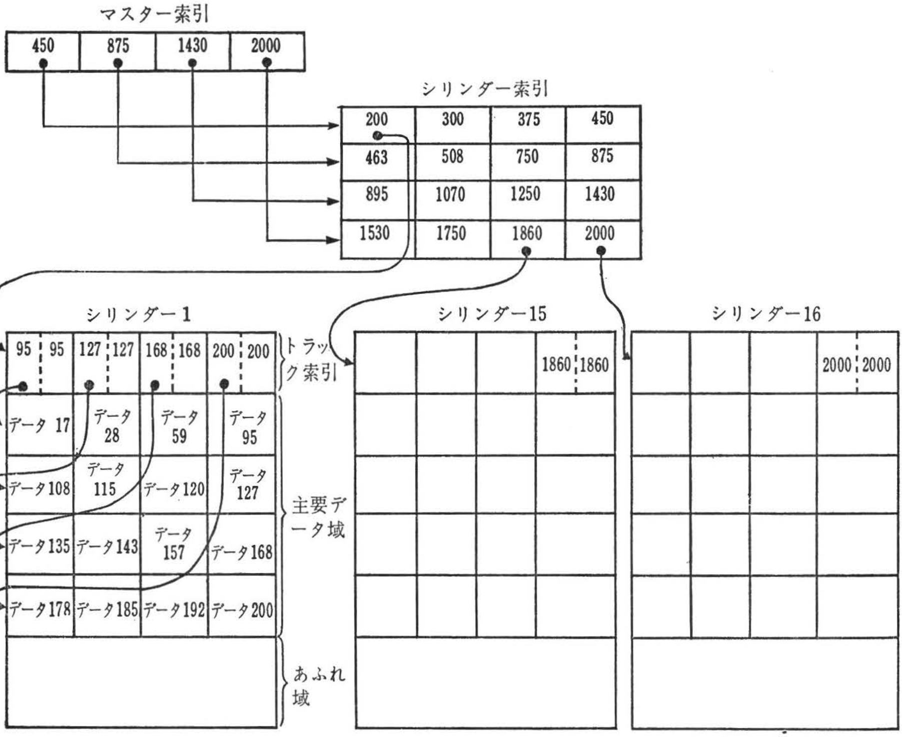


図 6-39　索引順次データ・セット編成の構造

めの情報が貯えられている．一方，あふれ域は，新規レコードの追加によって，主要データ域からあふれたレコードを収容するためのものであり，百科事典にはこれに相当するものは見当らない．

レコードが主要データ域に書き出されていくさい，オペレーティング・システムは必要に応じて各々のトラック上のレコードを探索するのに役立つ情報，つまり，トラックの最後に書き出されたレコードのキーとそのトラック・アドレスとを対にした項目を**トラック索引区域**（track index area）に入れる．トラック索引区域は各シリンダーの最初のトラックに設けられる．データ・レコードが2つ以上のシリンダーにまたがる場合だと，自動的にシリンダー索引を

作ってくれる．シリンダー索引内の**記入項目**（entry）は，シリンダー内の最後のレコードのキーとシリンダー・アドレスを対にしたものである．シリンダー索引の探索速度を高めるためには，図6-39でも示しているように1つレベルをあげた索引，つまりマスター索引を作るように要請することも可能だ．

1）主要データ域

データ・セットを作成したり更新すると，レコードは**主要データ域**（prime data area）に書かれる．主要データ域は，複数個の不連続の場所からなりたっていてもかまわないが，主要データ域に収容されるデータ・レコードはすべてキー番号の順に書き出されている必要がある．したがって，各レコードはキーを持っている必要がある．主要データ域でのレコードの形式は**図6-40**に示したように，必ずカウント／キー／データの形式である．非ブロック化レコードの場合には，キー部で各論理レコードのキー情報が示されるのでデータ部にキーを持っている必要はない．一方，ブロック化レコードだと，各論理レコード内にもキー・フィールドが必要である．なぜなら，キー部にはそのブロックの最後のレコードのキーしか入らないからである．索引順次データ・セットでは，不定長レコードは取扱えない．固定長か可変長レコードのいずれかの形式

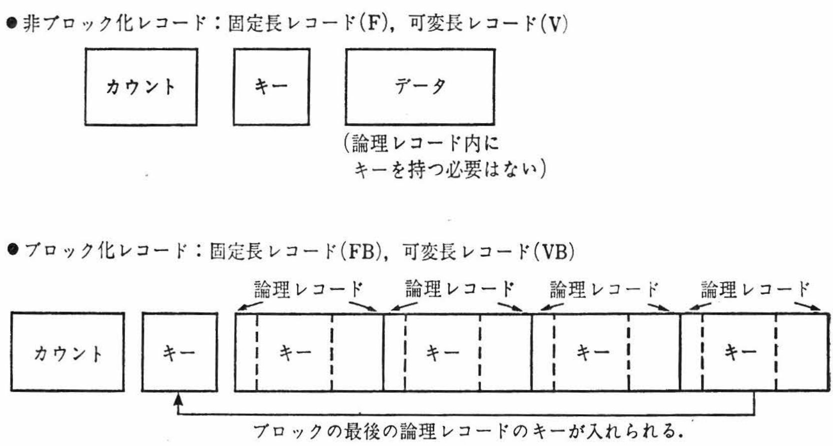


図6-40　索引順次データ・セットのレコード形式（主要データ域の場合）

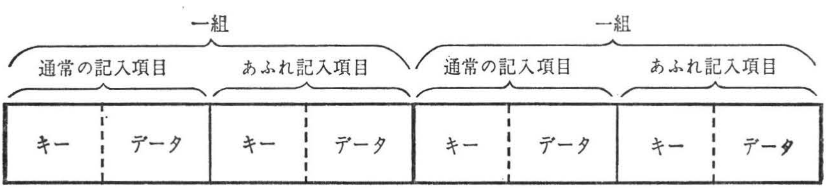
が取扱える．なお，主要データ域ではブロック化レコードとしても記憶できるが，あふれ域に入るレコードは必ず非ブロック化レコードになってしまう．

2) 索　引　域

トラック索引 (track index) と**シリンダー索引** (cylinder index) とは，オペレーティング・システムが自動的に作りあげてくれる．しかし，**マスター索引** (master index) は，プログラマーが DCB マクロ命令のパラメータ (OPTCD=M) を指定したときだけ作られる．

トラック索引は，主要データ域に割り当てられた各々のシリンダーの最初のトラックに作り出される．トラック索引は，シリンダー上の各トラックに対応して２個の記入項目を持つ．**通常の記入項目** (normal entry) と**あふれ記入項目** (overflow entry) である（図 6-41参照）．

通常の記入項目は，そのトラックに収容されているレコードの最も大きいキーと該当トラックのアドレスを対にしたものから構成される．一方，あふれ記入項目には，そのトラック上に最初に記録されたレコードのうちの最大のキーと該当トラックのアドレスを対にしたものが入る．つまり，最初の状態は通常の記入項目の内容と全く同じである．しかし，そのトラックからレコードあふれが生じると，あふれ域に移されたレコードのうちの最も小さいキーをもったレコードのアドレスが入れられる．



●通常の記入項目のキー：主要データ域のトラック上の最後のレコード・キー
●通常の記入項目のデータ：主要データ域のトラック・アドレス
●あふれ記入項目のキー：主要データ域のトラックと論理的関連を有するあふれレコードの最大のキー
●あふれ記入項目のデータ：主要データ域のトラックと論理的関連を有する最もキーの小さいあふれレコードのアドレス

（あふれ記入項目のキーとデータについて）あふれレコードがない場合には，通常の記入項目と同じデータとキーをもつ．

図 6-41　トラック索引の記入項目の形式

トラック索引に要するスペースは，キーの長さと各シリンダーの主要データ域のトラック数に依存する．トラック索引に用いられるトラックの余ったスペースは自動的に主要データ域の一部として利用される．なお，トラック索引の

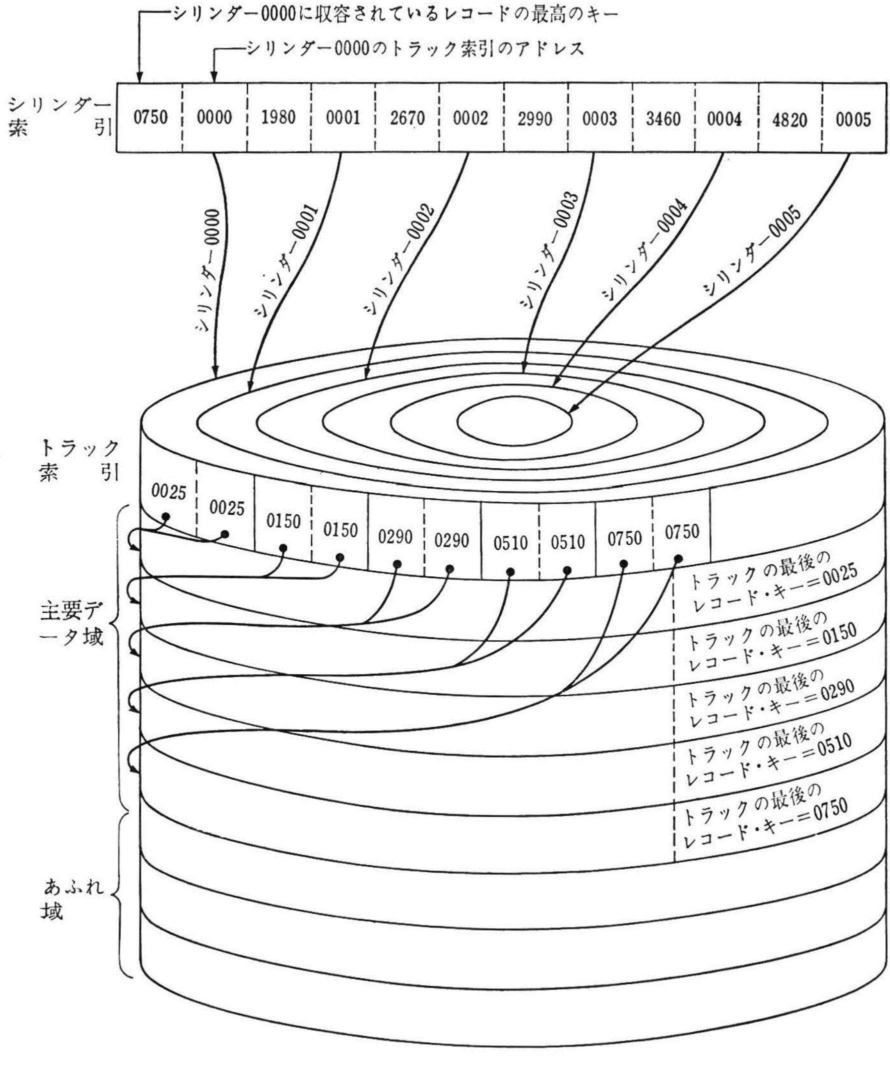


図 6-42　シリンダー索引とトラック索引

記入項目は，非ブロック化固定長のレコードでカウント／キー／データの形式になっている．これは，索引の探索をキー・サーチ機能を使って行えるようにしたものであり，探索効率を高めるためである．

各々のトラック索引に対応して１個のシリンダー索引の記入項目が作られる．１つの索引順次データ・セットには１個のシリンダー索引が用意され，それぞれの記入項目でトラック索引をポイントする（**図6-42**参照）．各記入項目には，該当シリンダーに収容されているレコードのうちの最大のキーとトラック索引のアドレスが入っている．

一方，マスター索引の記入項目は，シリンダー索引のトラックをポイントする．このマスター索引はシリンダー索引が数トラックにまたがるような大きさの場合，余分な探索を回避するために設けるものである．プログラマーは，マスター索引に入れるべき記入項目の最高数を指示できる．たとえば，シリンダー索引の記入項目が３トラック書かれるごとにマスター索引を作り出すように指定する．この場合には，シリンダー索引が３つのトラックをこえるときにマスター索引が作られる．データ・セットが非常に大きくて，マスター索引も３トラックをこえるとなると，自動的にそのマスター索引の１つ上のレベルの別のマスター索引を作ってくれるのである．このようにして，最高３レベルまでのマスター索引を作ってくれる．

3) あふれ域

索引順次データ・セットにレコードが追加されると，そのレコードのキーにもとづいて論理的に正しい位置に挿入される．その結果，該当トラックにもともと存在していたレコードが押し出されて，あふれ域に移されることになる．

図6-43を参照されたい．この図で，レコードが追加されるさいのレコードの動く様子を示している．キー25を持ったレコードが追加されると主要データ域の最初のトラックの所定の位置に挿入される．その結果，挿入レコードの位置以降に存在していたレコードはその位置がずらされてゆき，当該トラック上の最後に位置していたレコード(キー 100)が押し出されてしまい，あふれ域に移される．キー100を持ったレコードがあふれ域に移ったことにより，トラック

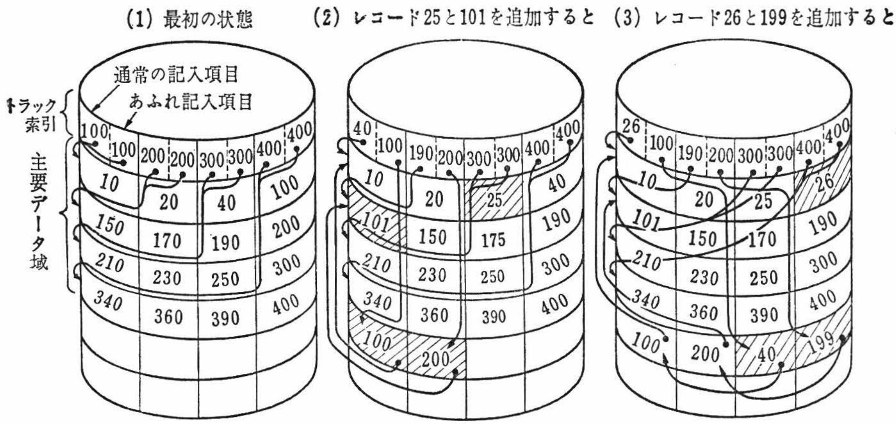


図 6-43　索引順次データ・セットへのレコードの追加

索引の記入項目が変ることに注目していただきたい．通常の記入項目のキーの部分は，対応するトラック上の最後のレコードが変ったことを反映して変化する．一方，あふれ記入項目のデータ部分のアドレスはあふれ域に移されたレコードを指すようになる．なお，あふれ域のレコードは必ず非ブロック化形式であり，各々のレコードの最初に10バイトの**余分のフィールド**（sequence link field）がつけられ，これによってあふれ域に移されたレコードの順序関係を示す．あるトラックからあふれたレコードのうち，もともとそのトラック上の最後のレコードであったものの**順次連鎖フィールド**（sequence link field）には元のトラックのアドレスが入る．このようにあふれ記入項目と順序連鎖フィールドを使って，レコードの論理的順序の関係づけを行っているのである．図6-43において，キー101を持ったレコードが追加される場合にも，主要データ域の2番目のトラック上のレコードに対して同様の手順の処理が加えられる．

同一トラックに2つ以上のレコードが追加されると，図6-43の右側の例のようになる．まず，キー26のレコードが追加された場合をみていただこう．キー40のレコードがあふれ域に移ると同時に，トラック索引が変化する．あふれ記入項目のデータ部分のアドレスは新たにあふれ域に移されたレコードをポイントする．また，あふれ域に移されたレコードの順序連鎖フィールドには，論理

的に次に続くべき別のあふれレコードをポイントするためのアドレスが入る．一方，キー199のレコードの追加の場合には若干異る．追加操作の最初には必ず索引を探索して，追加レコードがどのトラックに所属すべきかの判定が行われる．キー199の場合には，トラック索引のキー200と比較されることによって，はじめて小さいという判定がくだされる．ところでキー200を持っている記入項目はあふれ記入項目であることから，そのレコードは本来主要データ域の2番目のトラックに位置すべきであるにもかかわらず，そのトラックには論理的順序からして入れることができないことを意味する．したがって，はじめからあふれ域に移され，あふれ記入項目と順序連鎖フィールドとでレコードの前後関係がつけられるのである．

以上のような方法でレコードの追加が行われ，あふれ域が利用されるのであるが，あふれ域には2種類がある．**シリンダーあふれ域**（cylinder overflow area）と**独立あふれ域**（independent overflow area）とである（**図6-44**参照）.

シリンダーあふれ域とは，それぞれのシリンダーの終りに一定のトラック数のあふれ域を設ける方法である．あふれ域に割り当てるトラック数は各シリンダーとも同一数である．シリンダーあふれ域を用いると，各レコードを探し出す

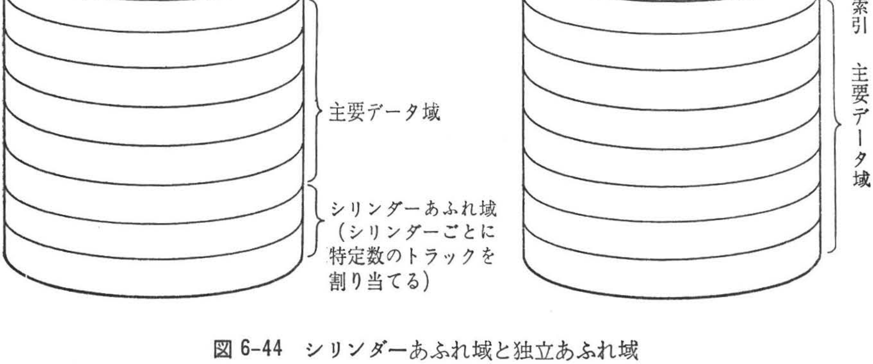


図6-44　シリンダーあふれ域と独立あふれ域

のに要する時間が少なくてすむ．しかし追加レコードがデータ・セットを通じて不均等に分布する場合，使用されない無駄のスペースが多くなる可能性がある．

独立あふれ域は，あふれレコードだけを収容する目的で数シリンダーを用いる方法である．このあふれ域は，主要データ域と同じボリューム上に設けてもよいし，異ったボリューム上に設けてもよい．独立あふれ域の場合には，あふれレコードのために確保するスペースの無駄を少なくすることができるが，あふれレコードを探し出すのに要する時間は増える．

シリンダーあふれ域も独立あふれ域も長所と短所がある．そこでシリンダーあふれ域と独立あふれ域の両者を組合せて用いる方法がある．この場合，シリンダーあふれ域には，平均的なあふれレコードを反映したトラック数を確保しておく．まず，シリンダーあふれ域が使われ，シリンダーあふれ域が一ぱいになったときだけ，独立あふれ域にレコードが入れられる．

2.　索引順次データ・セットの処理の仕方

索引順次データ・セットでは多様な処理が可能である．レコードを順番に処理することも，特定レコードを直接取り出して処理することもできる．レコードを追加することも，また更新することもできる．いずれの処理の場合にも，データ・セットを収容しているすべてのボリュームがオンライン状態であることが要求される．QISAM および BISAM を使った具体的な処理の方法を説明するまえに，順次処理，直接処理，追加，削除および再編成の仕方について概観しておこう．

1)　順次処理

レコードの取出しは，トラックからトラックへ，シリンダーからシリンダーへと進んでゆく．トラック索引が参照されるのは，各トラック上の最後のレコードを処理したあとだけであり，そのトラックに関連するあふれレコードがあるかどうかをあふれ記入項目の内容を調べて確かめる．もし，あふれレコードがあれば，あふれ記入項目にあるアドレスをもとに取り出しを続行する．あふれレコードを取り出すと，順序連鎖フィールドをもとに後続あふれレコードを

取り出してくれる．次のトラック上のレコードの取り出しに移るまえに，あふれ記入項目と順序連鎖フィールドを使って正しいキーの順番に取り出しを完成してくれるわけである．主要データ域にあるレコードは，少なくとも各トラックの範囲内ではキー順になっているので，順序データ・セットのレコードを取り出すのと全く同じ方法で順次処理が行える．

順次処理の場合，データ・セットの最初から最後まで全レコードを処理することもできるし，特定範囲内のレコードだけを順番に取り出して処理することもできる．たとえば，人事レコードが索引順次データ・セットの編成になっているとした場合，全社員のレコードを取り出して処理することもできるし，特定部門の社員レコードだけを順番に取り出して処理することも可能なのである．

なお，順次処理においては特定範囲の開始アドレスを探し出す場合を除いて，シリンダー索引とマスター索引を参照することはない．

2) 直接処理

特定レコードをランダムに取り出す場合には，そのレコードがどこに記憶されているかをつきとめるために，あらゆるレベルの索引を使う．マスター索引が設けられているならば，この索引から探索を開始する．特定レコードのキーをもとにマスター索引のキー・サーチを行う．等しいか小さい状態が生じると，その記入項目のアドレスをもとにシリンダー索引の探索をする．やはり，等しいか小さい状態が生じることによって，その記入項目のアドレスで該当トラック索引がつかめる．トラック索引をサーチしたあとで，特定レコードが記録されているトラックを見い出す．主要データ域にレコードが記録されている場合には，そのトラックをキー・サーチし，あふれ域に記録されている場合だと順序連鎖フィールドを使って物理レコードをつきとめる．そのあとで，物理レコードを主記憶装置に読み込み，必要とする論理レコードだけを処理プログラムに手渡してくれる．

なお，マスター索引だけは主記憶装置に読み込んでおいて，索引探索を行うことが許されている．したがって，キー・サーチよりも一段と高速で探索を行

うことが実現可能となる.

3) レコードの追加

索引順次データ・セットにレコードを追加する場合には，データ・セットの途中にレコードを挿入することも，既存のデータ・セットの最後のレコードの後続レコードとしてデータ・セットを拡大することも容易にできる.

データ・セットの途中にレコードを挿入する方法でレコードを追加するやり方は，図6-43で詳しく説明した通りである．これによって，キーの論理的順序にしたがって正しい位置に追加される．直接処理の場合と同じようにマスター索引，シリンダー索引およびトラック索引を活用して，所定の位置をつきとめ追加オペレーションが達成される．また，追加レコードを反映してトラック索引の記入項目も自動的に更新される.

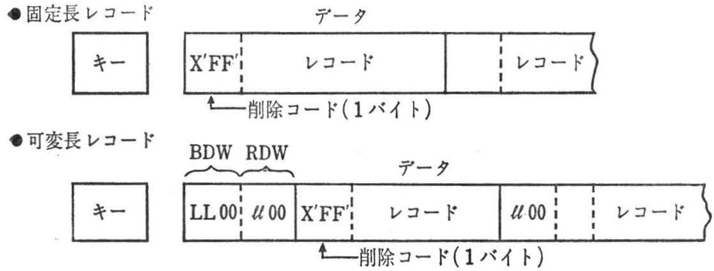
データ・セットの最後にレコードを追加するのは，追加レコードのすべてのキーが既存レコードの最後のレコード・キーよりも大きい場合である．したがって，主要データ域を拡大する方法で追加され，その追加に応じたトラック索引やシリンダー索引が作り出される.

なお，前者の追加にはBISAMが，後者の追加にはQISAMが適用される.

4) レコードの削除

索引順次データ・セットの削除レコードは，通常のオペレーションでは物理的には削除されない．追加オペレーションや再編成のさいにはじめて物理的な削除が達成されるのである．したがって，削除レコードは単に不要レコードである旨の目印をつけておく．**図6-45**の上段で示しているようにレコードの直前に余分の1バイトを設けておき，削除レコードに対してはこの場所にすべて1ビットを入れる．この作業は，更新処理を通じて行える.

このような目印のついたレコードは，その後の順次処理においては全く取り出されないので処理の対象から除外されてしまう．しかし，直接処理の場合には，他のレコードと同じように取り出されてしまうので処理プログラムの中で削除コードの検査を行うことが要求される．なお，挿入の方法で追加が行われ



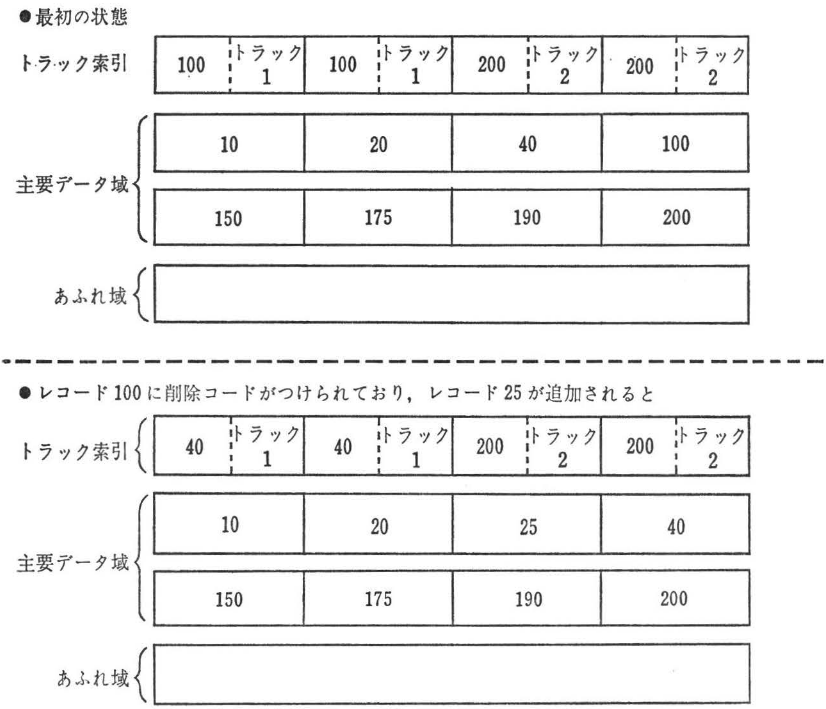


図 6-45　索引順次データ・セットからのレコードの削除

ると，図 6-45の下側で図示したように削除コードのついたレコードがトラックの終りからおし出されると，そのレコードはもはやあふれ域に書かれることはなく，物理的に削除されてしまう．

5）データ・セットの再編成

索引順次ファイルは，つぎのような理由で定期的に再編成を行う必要がある．あふれ域や索引域がレコードの追加によっていっぱいになったり，削除コードのついたレコードが増えることによってデータの取り出しに余分の時間がかかったり，スペースの無駄が生じるためである．再編成を行う頻度は，データ・セットの利用具合いあるいは時間的・記憶容量的な諸条件を勘案してきめる必要がある，データ・セットの再編成を行う方法として，つぎの2通りがある：

- データ・セットをいったん DASD 上の別の記憶域か磁気テープ・ボリューム上に順次データ・セットとして書き出したうえで，再び元の場所に書きもどす．
- DASD 上の別の記憶域に索引順次データ・セットとして書き出すことによって，直接再編成を達成する．この方法だと，旧データ・セットで占めていた記憶域を再編成データ・セットで利用することはできない．

なお，オペレーティング・システムは再編成に関する各種統計をとってくれる．その情報としては，シリンダーあふれ域の数，独立あふれ域の未使用トラック数，あふれレコードの参照回数などである．これらの情報は DASD 上に書き出されるが，ユーザーが利用できるように DCB の特定フィールドにもどされる．

3. QISAM による索引順次データ・セットの処理

QISAM の主要な機能としては，つぎの5つのものがある：

- 索引順次データ・セットを新規に作成する機能
- 既存の索引順次データ・セットの最後に後続レコードを追加して，データ・セットを拡大する機能
- 索引順次データ・セットのレコードを順番に処理する機能．つまり，データ・セットの順次処理機能．

- 索引順次データ・セットの特定グループないしは特定範囲のレコードを順次処理する機能
- 索引順次データ・セットを順次的に更新する機能.

QISAM で使用する GET/PUT マクロ命令は, QSAM のそれとほとんど変りない. 移動方式も位置指定方式もある. 一方, DCB マクロ命令のパラメータはこれまで説明してきた QSAM, BSAM あるいは BPAM のそれと共通するものも多いが, QISAM 特有のパラメータもいくつかある. **表 6-6** に索引順次データ・セットの DCB マクロ命令で使用される主要なパラメータの一覧表を示しておいたので, そのつど参照されたい. この一覧表では QISAM だけでなく BISAM の場合のパラメータもまとめて示してある. 多くのパラメータは, 索引順次データ・セットの作成時だけに指定すればすむ. パラメータで指定した内容がデータ・セット・ラベルに収容されるからである. したがって, いったん作成したデータ・セットを利用する場合には, 必要な情報をデータ・セット・ラベルから得ることができるので, もはや DCB マクロ命令で指定する必要はなくなるのである. なお, MACRF パラメータで指定する記号は, 若干面倒くさいので別表として**表 6-7** にまとめてある. 表 6-6 と一緒に必要のつど参照されたい.

パラメータ	摘要		指定方法
	BISAM	QISAM	
BLKSIZE=	×	データ・セットの作成時のみ	物理レコードのバイト数の指定.
CYLOFL=	×	データ・セットの作成時のみ	シリンダーあふれ域に割当てるトラック数
DDNAME=	○	○	データ定義名を指定.
DSORG=IS または ISU	ISだけ	○	IS (Indexed Sequential) または ISU (Indexed Sequential Unmovable) を指定.
EODAD=	×	○	データ・セットの最後まで処理したときのブランチ・アドレス(入力の場合だけ)
KEYLEN=	×	データ・セットの作成時のみ	キーの長さをバイト数で指定.

LRECL=	×	データセットの作成時のみ	論理レコードの長さをバイト数で指定.
MACRF=	○	○	第7表参照のこと.
MSHI=	○	×	最高レベルの索引（マスター索引）を貯える主記憶域のアドレス.
MSWA=	○	×	作業域として制御プログラムが使用する主記憶域のアドレス.
NTM=	×	データ・セットの作成時のみ	最低レベルのマスター索引の各記入項目に対応するシリンダー索引のトラック数
OPTCD=(W, M, Y, I, L, R, U)	×	データセットの作成時のみ	I：独立あふれ域を要求. L：削除コードをもったレコードの削除を要求（追加のさいに）. M：マスター索引の作成を要求. R：再編成に必要な統計情報を要求. U：トラック索引の記入項目をそのつど書き出さないで，トラック索引の各トラックに対してグループとして書き出すことを要求. W：書き出しオペレーションにおいて妥当性チェック（validity check）を要求. Y：シリンダーあふれ域を要求.
RECFM={F, FB, V, VB}	×	データ・セットの作成時のみ	使用するレコード形式を指定.
RKP=	×	データ・セットの作成時のみ	レコード・キーの最初のバイトの位置，つまり各レコードの最初からの相対的位置
SMSI=	○	×	最高レベルの索引に要する主記憶域のサイズ．MSHI パラメータと対で指定.
SMSW=	○	×	作業域に要する主記憶域のサイズ．MS-WAパラメータと対で指定.
SYNAD=	○	○	入出力エラー分析ルーチンのアドレスを指定.

表 6-6　QISAM および BISAM の DCB マクロ命令で使われる主要なパラメータ

1）索引順次データ・セットの作成

QISAM のとき. $MACRF=\left\{\begin{array}{l}(PM)\\(PL)\\(GM\,[,S\left[\begin{array}{c}K\\I\end{array}\right]])\\(GL\,[,S\left[\begin{array}{c}K\\I\end{array}\right]]\,[,PU])\end{array}\right\}$	GL：位置指定方式の GET マクロ命令を使用する場合. GM：移動方式の GET マクロ命令を使用する場合. I：装置アドレスを使って探索を行う場合. K：レコード・キーを使って探索を行う場合. PL：位置指定方式の PUT マクロ命令を使用する場合. PM：移動方式の PUT マクロ命令を使用する場合. PU：PUTX マクロ命令を使ってレコードの更新をする場合. S：SETL マクロ命令を使う場合.
BISAM のとき. $MACRF=\left\{\begin{array}{l}(R[S][C])\\(W\left\{\begin{array}{l}U\\A\\UA\end{array}\right\}[C])\\(R\left[\begin{array}{l}S\\U\\US\end{array}\right][C],W\left\{\begin{array}{l}T\\A\\UA\end{array}\right\}[C])\end{array}\right\}$	A：レコードを追加する場合. WRITEマクロ命令で KN を指定のこと. C：CHECH マクロ命令を使用する場合. R：READ マクロ命令を使用する場合. S：動的バッファ割当てを要求する場合. U：レコードの更新を行う場合. W：WRITE マクロ命令を使用する場合.

表 6-7 MACRF パラメータの指定方法

索引順次データー・セットを作成する場合の手順を，**図6-46**に示したサンプルを参照しながら説明してゆこう.

ジョブ制御カードの DD ステートメントでは DSNAME でデータ・セット名とかっこ内で主要データ域である旨を PRIME と指示する. このように主要データ域だけを要求すると，必要に応じてその区域の中からマスター索引とシリンダー索引のスペースをとってくれる. また，索引のために使用した最後のシリンダーの未使用スペースを独立あふれ域として割り当ててくれる. ところで，索引のために主要データ域とは別個のスペースを使いたい場合には，もう一枚別の DD ステートメントが必要である. そして，データ・セット名の直後にかっこで INDEX と指示する. そのうえで，SPACE と UNIT パラメータ

DD ステートメント

```
//ISDD   DD      DSNAME=MASTER.INDEX6(PRIME),DCB=DSORG=IS, C
//               UNIT=2311,SPACE=(CYL,(25,,2),CONTIG),     C
//               DISP=(,KEEP)
//INDD   DD      ……
```

処理プログラム

```
              :
         OPEN    (INDATA,,ISDATA,(OUTPUT))
NEXT     GET     INDATA
         LR      0,1
         PUT     ISDATA,(0)
         B       NEXT
ERROR     |
          ↓      出力エラー処理
ENDRTN   CLOSE   (INDATA,,ISDATA)
              :
INDATA   DCB     DSORG=PS,MACRF=(GL),EODAD=ENDRTN,              C
                 DDNAME=INDD
ISDATA   DCB     DSORG=IS,MACRF=(PM),RECFM=FB,LRECL=100,        C
                 BLKSIZE=1100,KEYLEN=10,RKP=1,                  C
                 OPTCD=ILMWYR,                                  C
                 CYLOFL=1,NTM=3,BUFNO=4,DDNAME=ISDD,            C
                 SYNAD=ERROR
              :
```

図 6-46　索引順次データ・セットの作成

で索引に要するスペースと装置を指示する．なお，データ・セット名は同じにし，次の順序で指定する：

```
// ddname DD DSNAME=dsname (INDEX),…
//        DD DSNAME=dsname(PRIME),…
```

また，独立あふれ域を別個に設ける場合にも，索引域を別に設ける場合と本質的には同じ方法をとる．ただ，INDEX の代りに OVFLOW とする．したがって索引域も独立あふれ域も別個に設けるとしたら，次のようになる：

```
// ddname DD DSNAME=dsname(INDEX),…
```

```
//          DD DSNAME=dsname(PRIME), …
//          DD DSNAME=dsname (OVFLOW), …
```

話をもとにもどそう. DD ステートメントでは, DCB パラメータで DSORG=IS（または ISU）を必ず指定する. また, UNIT パラメータで使用する装置名を, SPACE パラメータを使って必要なスペースを指定する. さらに, DCB パラメータでレコードの長さ（LRECL), ブロック・サイズ（BLKSIZE), レコード形式（RECFM), キーの長さ（KEYLEN), 相対的なキーの位置（RKP), 必要な任意機能（OPTCD), シリンダーあふれ域（CYLOFL), マスター索引のトラック数（NTM）など DCB の属性を指定してもよい. 図6-46の例では, DCB マクロ命令でこれらの属性を指示しているので, DD ステートメントの DCB パラメータでは指定していない.

処理プログラムの DCB マクロ命令のオペランドでは, DSORG=IS または ISU および MACRF=(PM) または (PL) を指定する. その他, 例に示したようなパラメータを必要なだけ指定する.

出力のためのデータ・セットを OPEN したあとで, PUT マクロ命令を用いて, すべてのレコードまたはブロックを DASD 上に書き出す. すべての出力が終ったら, 例によってデータ・セットを CLOSE する. これで索引順次データ・セットが作成できるのである. なお, 新しいデータ・セットを構成するために入力されるレコードは, 必ずキーの順番にあらかじめ分類されている必要がある.

このようにして作成した索引順次データ・セットに新規レコードを追加してデータ・セットを拡大することは簡単に行える. 基本的には, データ・セットの作成手順と同じである. ただし, DCB マクロ命令でもはやデータ・セットの属性を指定する必要はない. データ・セット作成時にデータ・セット・ラベルにそれら属性が記録されているからである. OPEN マクロ命令が実行されると, これら属性情報が DCB を満してくれる. また, DD ステートメントの SPACE パラメータも不要である. DISP パラメータは NEW ではなく MOD を指定しておくことが望まれる. なお, データ・セット作成用のプログラムで

追加作業をも行わせるようにするためには，作成用プログラムの中の DCB マクロ命令でデータ・セット属性を指定することを避ける方法をとればよい．つまり，これら属性を DD ステートメントの DCB パラメータで指定するようにしておけば，DD ステートメントを取りかえるだけで同じプログラムが追加作業にも使えることになる．

2)　索引順次データ・セットの順次取出し

索引順次データ・セットの順次取出しの開始点を指定するためには SETL マクロ命令を使用する．SETL マクロ命令を用いると，データ・セットの最初から，あるいはデータ・セットの任意の場所からレコードの取出しを開始することができる．データ・セットの最初から処理する時以外は，レコード・キー，キー・クラス，あるいは実アドレスで処理の開始点を指定する．

レコード・キーの方法だと処理すべき最初のレコードのキー全体を正しく指定する必要がある．これに対して，キー・クラスの場合だとレコードのグループを表示する最初の何桁かを正確に指示し，残りの桁は2進数のゼロで指示する．この指定方法は，プログラマーが最初に処理すべきレコードのキー全体を知らなくてもよいという点で極めて好都合である．実アドレスを用いるために

```
         ⋮
         OPEN    (ISDATA)
         SETL    ISDATA, B
NEXT     GET     ISDATA
          ↓      レコードの処理
         B       NEXT
ERROR     ↓      エラー処理ルーチン
ENDRTN   CLOSE   (ISDATA)
         ⋮
ISDATA   DCB     DDNAME=INDEXDD, DSORG=IS, MACRF=(GL, S),   C
                 SYNAD=ERROR, EODAD=ENDRTN
         ⋮
```

図 6-47　索引順次データ・セットの順次取出し (1)

は，データ・セットを作り出すときレコードが書き出された場所を記録しておかなければ使えないという不便さがある．

GET マクロ命令でのレコードの取出しは，データ・セットの終りあるいはCLOSE マクロ命令か ESETL マクロ命令が実行されるまで続く．ESETL マクロ命令は索引順次データ・セットからのレコードの取り出しをストップするようシステムに指示する．ESETL マクロ命令を実行したら，別の新しい範囲の取り出しを開始させるために再び SETL マクロ命令を実行させることもできるし，そこで CLOSE マクロ命令を与えて処理を終らせることもできる．

索引順次データ・セットのレコード全体を順次取出しする場合の例が，**図6-47**である．DCB マクロ命令の MACRF パラメータでは位置指定方式の GET マクロ命令（GL）と SETL マクロ命令（S）を使用する旨の指定をする．データ・セットを OPEN したあとで，データ・セットの最初のレコードから取出しを行うことを SETL マクロ命令で指示する．SETL マクロ命令の第２オペランドでＢと指定することによってデータ・セットの最初のレコードから取出しが可能となる．なお，BD と指定すると非ブロック化レコードの場合，データ・フィールドの部分だけの取出しを意味する．Ｂだとキー・フィールドを含めての取出しである．実際のレコードの取出しは，GET マクロ命令で行う．例では，位置指定方式の命令を使っているが，移動方式の GET マク命令を使うこともできる．データ・セットの最後のレコードの取出しが終ったら，CLOSE マクロ命令を与えればよい．

図6-48では，特定レコード・グループの順次取出しの例を示している．入力カードで索引順次データ・セットから取出すべきレコード・グループの下限と上限のキーを示しているものとする．取出されたレコードの特定フィールドを印刷する仕事である．

索引順次データ・セットの DCB マクロ命令の MACRF パラメータでは，移動方式の GET マクロ命令（GM）とレコード・キーによる SETL マクロ命令（SK）を使用することを指示する．データ・セットを OPEN したあと，特定レコード・グループの取出し下限のキーと上限のキーを得るためにカードを読み取る．SETL マクロ命令で入力カードで得た取出しレコードの開始点を

```
                 ⋮
         OPEN    (INDATA,,ISDATA,,PRINT (OUTPUT))
GETCARD  GET     INDATA,CDWORK
         SETL    ISDATA,K,LOWKEY
GETIS    GET     ISDATA,ISWORK
           ↓     レコードの処理
         PUT     PRINT,PLINE
         CLC     ISKEY,HIGHKEY     グループの最後のレコードかテスト
         BL      GETIS
         ESETL   ISDATA
         B       GETCARD
LAST     CLOSE   (INDATA,,ISDATA,,PRINT)
                 ⋮
INDATA   DCB     DSORG=IS,MACRF=(GM),EODAD=LAST,…
ISDATA   DCB     DSORG=IS,MACRF=(GM,SK),…
PRINT    DCB     DSORG=PS,MACRF=(PM),…
*
CDWORK   DS      0CL80
LOWKEY   DS      CL8
HIGHKEY  DS      CL8
         DS      CL64
PLINE    DS      CL133
ISWORK   DS      CL100
ISKEY    DS      CL8
REST     DS      CL92
                 ⋮
```

図 6-48　索引順次データ・セットの順次取出し (2)

指示する．そのために，SETL マクロ命令の第2オペランドでレコード・キーを使用する旨（K）を指定し，第3オペランドでそのキーが入っている場所を指定する．もし，キー・クラスを使用するのであれば第2オペランドで KC と指定する．実アドレスを使用するのなら，I と指定する．ともあれ，このマクロ命令で開始点のレコード・キーをセットしたあとで GET マクロ命令でレコードの取出しを行う．印刷処理を行ってから，グループの最後のレコードまで処理したかどうかを比較命令でテストし，特定グループの最後まで処理した場合には，ESETL マクロ命令でそのグループの処理を終らせて，別のレコー

ド・グループの処理に移る．

以上のように索引順次データ・セットの取出しでは，データ・セット全体のレコードをもれなく取出すことも出来るし，特定レコード・グループを自由に選んで，その範囲内の順次取出しを行うこともできるのである．しかも，特定範囲の取出しでは，レコード・キー，キー・クラスあるいは実アドレスで取出しレコードの開始点が指定できる．

3) 索引順次データ・セットの順次更新

図 6-49で索引順次データ・セットの順次更新の例を示している．この例では，キー・クラスが915である全レコードを順次取出しして，各レコードの13

```
          ⋮
          OPEN    (ISDATA)
          SETL    ISDATA, KC, KEYADDR    取出しの下限を設定
          TIME                           レジスター 1に今日の日付けを
          ST      1, TODAY
NEXT      GET     ISDATA
          CLC     19(10, 1), LIMIT       レコード・キーと上限を比較
          BNL     ENDJOB
          CP      12(4, 1), TODAY        レコードの日付け（有効期日）と今日の
          BNL     NEXT                   日付けを比較
          MVI     0(1), X'FF'            有効期日を越えたレコードに削除コードを
                                         入れる．
          PUTX    ISDATA                 削除レコードを書きもどす．
          B       NEXT
ERROR     ↓
                  エラー分析ルーチン
ENDJOB    CLOSE   (ISDATA)
          ⋮
TODAY     DS      F
KEYADDR   DC      C'915'                 キー・クラス
          DC      XL7'0'                 残りのキー部分（2進数のゼロ）
LIMIT     DC      C'9160000000'          上限のキー
ISDATA    DCB     DDNAME=INDEXDD, DSORG=IS,                      C
                  MACRF=(GL, SK, PU), SYNAD=ERROR
          ⋮
```

図 6-49 索引順次データ・セットの順次更新

から16バイト目に入っている日付がプログラムの実行日の日付（TIME マクロ命令を与えることによりレジスター1に 00yyddds の形式で得られる）より前の日付の場合には，レコードの最初のバイトに削除コード（X'FF'）を入れたうえで，書きもどしている．この例にもとづいて，索引順次データ・セットの順次更新の手順を説明しよう．

DCB マクロ命令の MACRF パラメータでは，移動方式の GET マクロ命令（GL）を使ってレコードの更新を行い，PUTX マクロ命令（PU）で更新レコードを書き出すことを指示する．同時に，キー・クラスの SETL マクロ命令（SK）を使用することを示す．更新用データ・セットの OPEN マクロ命令ではオプションは不要である．したがって，QSAM のように UPDAT オプションは指定しない．

SETL マクロ命令で順次取出しの開始点をセットする．例では，キーの上位3桁が 915 であるレコードの最初から処理することを要求するために KC（キー・クラス）を第2オペランドで指定している．そのあとで，GET マクロ命令でレコードを取り出す．更新を行うための GET マクロ命令は必ず位置指定方式である．移動方式のそれは適用できない．

レコードを読みとったあと，レコード・グループの上限を越えているかどうか比較命令でテストし，範囲内のレコードであると削除コードを必要とするレコードかどうかテストする．更新の不要なレコードには PUTX マクロ命令を与えない．削除コードを必要とするレコードには，レコードの先頭のバイトにそのコードを入れたあと PUTX マクロ命令を与える．

レコード・グループの上限を越えると，順次取出しを終らせるために ESE-TL ママクロ命令を与えたあとで，次の特定グループの処理のために SETL マクロ命令に移る．しかし，例では1つのレコード・グループしか処理対象にしていないので，上限を越えたら直ちに CLOSE マクロ命令を与えて順次取出しを終えている．

このような手順で，索引順次データ・セットの順次更新が達成されるのである．

〔記号名〕	READ WRITE	DECB名, タイプ, DCBアドレス, {バッファ・アドレス / 'S'}, {長さ / 'S'}, キー・アドレス

●DECB名……DECB に割当てる記号名.

●タイプ ……READ/WRITE マクロ命令のタイプ. つぎのいづれかのコードを記入する.

●K ……READ の場合は通常の取出し. WRITE の場合は, 更新レコードの書き出し.

●KU……更新後にレコードが書き戻されることを指示. 制御プログラムはアドレスを保存する. (READ の場合のみ指定可能)

●KN……通常のレコードの書き出し (WRITE の場合だけ指定可能).

●DCBアドレス……読み書きするデータ制御ブロック (DCB) のアドレス.

●バッファ・アドレス……入出力ブロックを収容するための主記憶域のアドレス. 最初16バイトは制御プログラムが用いるのでユーザー・データは貯えられない. アドレスの代りに 'S' を指定すると**動的バッファ割当て** (dynamic buffering) を要求することになる. この場合, 確保された主記憶域のアドレスは DECB にもどされる.

●長さ……読み書きするデータ・バイト数.
入力の場合, 'S' を指定するとバイト数はレコードのカウント・フィールドから得られる. ブロック化レコードの場合は必ず 'S' とする.
出力の場合, 'S' を指定すると制御プログラムにすでに知らされている長さが用いられる. レコードの追加や更新の場合には必ず 'S' を指定する.

●キー・アドレス……キーを貯えている主記憶域のアドレス.
入力の場合, 取り出すべき論理レコードのキーを入れておく. 入力オペレーションが終ると論理レコードのアドレスは DECB に置かれる.
出力の場合, 新しいレコードあるいは更新レコードのキーを入れておく.

表 6-8 BISAM の READ/WRITE マクロ命令の一般形式

4. BISAM による索引順次データ・セットの処理

BISAM を使って, QISAM と同じ処理はできない. QISAM とは異った機能を受け持つのが BISAM である. QISAM の基本的機能が順次処理であるのに対して, BISAM の基本的機能は直接処理である. したがって, BISAM の主要な機能をまとめると, 次の 3 つになる：

●索引順次データ・セットのレコードをランダムに取り出す機能.
●索引順次データ・セットに追加レコードを挿入する機能.
●索引順次データ・セットのレコードをランダムに更新する機能.

いずれの場合にも，キーの順番に処理する必要はない．すべて直接処理である.

BISAM でも，レコードの読み書きには READ/WRITE マクロ命令を使用する．これらマクロ命令の形式は**表 6-8** で示している．BSAM や BPAM のマクロ命令の形式とあまり変らない．キー・アドレスが新たなオペランドとして追加されていることと，4番目のオペランドで動的バッファ割当て（'S'）の指定ができることが大きな差である．後述する BISAM のサンプル・コーディングを検討するさいに，この一般形式を参照されたい．また，DCB マクロ命令のパラメータは表 6-6 と表 6-7 に QISAM と一緒にまとめてあるので，これも必要のつど参照されたい.

なお，BISAM においてはレコードの処理およびエラー分析においてしばしば **DECB** (Data Event Control Block) の各フィールドを参照する．そのために，READ/WRITE マクロ命令によって生成される DECB の形式を示したのが**図6-50**である．また，**図6-51**において DECB がどのように生成されるかを READ マクロ命令を例にあわせて示しておいた．READ あるいは WRITE

+0	**事象制御ブロック** (Event Control Block : ECB) (4バイト)	
+4	オペレーションのタイプ (2バイト)	入出力データのバイト数 (2バイト)
+8	DCB のアドレス　(4バイト)	
+12	バッファのアドレス　(4バイト)	
+16	論理レコードのアドレス　(4バイト)	
+20	キーのアドレス　(4バイト)	
+24	例外状態コード　(2バイト)	

図 6-50　BISAM の DECB の形式

```
                :
            READ   DECB04, K, ISDCB04, INREC4, 'S', KEY04
+           CNOP   0, 4
+           BAL    1, *+30   DECBの次の命令にブランチ
+ DECB04    DC     A(0)      Event Control Block
+           DC     BL1'10'   タイプ・フィールド
+           DC     BL1'10000000'   タイプ・フィールド
+           DC     AL2(0)    入出力データのバイト数
+           DC     A(ISDCB04)    DCBアドレス
+           DC     A(INREC4)     バッファ・アドレス
+           DC     A(0)          論理レコードのアドレス
+           DC     A(KEY04)      キー・アドレス
+           DC     AL2(0)        例外状態コード
+           L      15, ISDCB04+88   RDWRT ルーチンのアドレスをロード
+           BALR   14, 15    RDWRT ルーチンへブランチ
```

マクロ・エクスパンション（CNOP から BALR までの行）

図 6-51 READ マクロのエクスパンション例

マクロ命令の第1オペランドで指定する DECB 名が DECB の最初のバイトにつけられる．したがって，図6-51の例で，論理レコードのアドレスを得るには DECB04+16 と指定すればよいし，例外状態コードを得るには DECB04+24と指定すればよい．後述のサンプル・コーディングで DECB のフィールドを参照する例があるが，その場合には図 6-50および図 6-51を想起していただきたい．

1） 索引順次データ・セットの直接取出し

図 6-52で索引順次データ・セットの直接取出しの例を示している．この例は，プログラマーガ READ マクロ命令でバッファを指定し，自分でバッファを確保する場合である．

DCB マクロ命令の MACRF パラメータでは，READ マクロ命令と CHECK マクロ命令を使用することを RC で指定する． CHECK マクロ命令ではなく WAIT マクロ命令を用いる場合は単にRだけでよい．なお，MSHI=MAST-INDX と SMSI=1000 の２つのパラメータを指定することにより処理速度をあげようとしている．この指定により，上位レベルのマスター索引は，プログ

```
          :
          OPEN      (ISDCB)
SETKEY    ↓         KELOC の場所に取り出すべきレコードのキーを挿入
          READ      MYDECB, K, ISDCB, INBUF, 'S', KEYLOC
          CHECK     MYDECB, DSORG=IS
          L         3, MYDECB+16  論理レコードのアドレスをレジスター
          ↓         取出しレコードの処理                         3へ
          B         SETKEY
ERROR     ↓
LAST      CLOSE     (ISDCB)
          :
MASTINDX  DS        CL1000
KEYLOC    DS        CL8
INBUF     DS        2D      制御プログラムが用いる16バイト
DATA      DS        CL500
ISDCB     DCB       DSORG=IS, MACRF=(RC), SYNAD=ERROR,           C
                    DDNAME=INDD, MSHI=MASTINDX, SMSI=1000
          :
```

図 6-52　索引順次データ・セットの直接取出し（プログラマーがバッファを割り当てる場合）

ラムの実行中主記憶域に常駐してくれる．その結果，DASD 上の索引を探索する必要がなくなるのでそれだけ処理効率があがる．しかし，余分な主記憶域をとってしまうのも事実である．例では1000バイトの場所をとり，その記憶域に MASTINDX という記号をつけている．

データ・セットを OPEN したあとで，直接取出しを行うべきレコードのキーを準備し，READ マクロ命令の6番目のオペランドで指定した記憶域に挿入する．そのあとで READ マクロ命令を実行する．READ マクロ命令の第2オペランドでは単にKを指定し，レコードの更新は行わないことを示す．6番目のパラメータとして指定した KEYLOC の場所に取り出すべきレコードのキーが入っている必要がある．CHECK マクロ命令の第2オペランドでは，特に DSORG=IS を指定する必要がある．

READ マクロ命令で入力された論理レコードが貯えられている場所のアレ

```
                :
              OPEN      (ISDCB)
SETKEY          ↓
              READ      MYDECB, K, ISDCB, 'S', 'S', KEYLOC
              CHECK     MYDECB, DSORG=IS
              L         3, MYDECB+16
                ↓
              B         SETKEY
ERROR           ↓
LAST          CLOSE     (ISDCB)
                :
MASTINDX      DS        CL1000
KEYLOC        DS        CL8
INDCB         DCB       DSORG=IS, MACRF=(RSC), SYNAD=ERROR,        C
                        DDNAME=INDD, MSHI=MASTINDX, SMSI=1000
                :
```

図 6-53　索引順次データ・セットの直接取出し（動的バッファ割当ての場合）

スは，DECB のフィールドに返される．したがって，そのアドレスを基点にレコードの各フィールドの処理を行う．そのために，論理レコードのアドレスをレジスターにロードしたうえで，レコードの処理を行う．

図 6-53もレコードの直接取出しの例であるが，**動的バッファ割当て**（dynamic buffering）を要求している．BISAM では制御プログラムにバッファ・エリアを自動的に割当てるようにさせることができる．制御プログラムが割当てたバッファ・エリアのアドレスは DECB の12番目のバイトから始まるフィールドに入れられる．ともあれ，動的バッファ割当てを要求するには，DCBマクロ命令の MACRF パラメータでSを指定する．READ マクロ命令と CHECK マクロ命令を使い，さらに動的バッファを要求するために MACRF=(RSC) となる．また，READ マクロ命令の4番目のオペランドでは，バッファ・エリアのアドレスを指示する代りに 'S' とする．これにより，READ マクロ命令が実行されるとき，まずバッファ・エリアの割当てが行われ，割当てられたエリアのアドレスが DECB に戻されるのである．

2) 索引順次データ・セットへのレコードの追加挿入

索引順次データ・セットの途中に新規レコードを追加挿入する例が，**図 6-54**である．DCB マクロ命令の MACRF パラメータでは，WRITE マクロ命令を使ってレコードを追加する旨指示する．さらに CHECK マクロ命令を使用することを要求する．したがって，MACRF=(WAC) となる．

WRITE マクロ命令を与えるに先立って，追加レコードを出力レコード・エリアに組立てると同時に，追加すべきレコードのキーを WRITE マクロ命令の6番目のオペランドで指定したエリアに入れておく．追加のための WRITE マクロ命令の第2オペランドでは必ず KN と指定する．この WRITE マクロ命令が実行されると，アクセス方式ルーチンでは指示されたレコード・キーにもとづいてマスター索引（もしあれば），シリンダー索引，トラック索引へと探索を続け，追加レコードの正しい挿入位置を見出して，その場所にレコードを追加する．同時に追加によって影響をうけるレコードの記憶位置を調整し，あふれレコードをあふれ域に収容し，トラック索引の更新を行ってくれるわけで

```
                :
        OPEN    (ISDCB, (OUTPUT))
SETUP    ↓
                追加レコードの組立ておよびレコードのキーを KEYLOC へ
        WRITE   MYDECB, KN, ISDCB, OUTBUF, 'S', KEYLOC
        CHECK   MYDECB, DSORG=IS
        B       SETUP
ERROR    ↓
LAST    CLOSE   (ISDCB, (OUTPUT))
                :
KEYLOC  DS      CL8
OUTBUF  DS      2D
OUTREC  DS      CL200
ISDCB   DCB     DSORG=IS, MACRF=(WAC), SYNAD=ERROR,        C
                DDNAME=OUT
                :
```

図 6-54　索引順次データ・セットへのレコードの追加挿入

ある.

なお，追加オペレーションを効率的に行うためには，追加オペレーションで使用する余分の作業域を DCB マクロ命令の MSWA パラメータと SMSW パラメータで指定すればよい．MSWA では提供する作業域のアドレスを，SMSW ではその作業域の大きさを指示する．もちろん，処理プログラムで作業域を確保しておく必要がある．こうしておくことによって，制御プログラムでは作業域を活用して，追加オペレーションの処理効率を高めてくれるのである.

3） 索引順次データ・セットの直接更新

索引順次データ・セットの直接更新には２つのやり方がある．その１つは，READ/WRITE マクロ形式として，**リスト実行形式** (list and execute form) を適用する方法であり， READ マクロ命令の第２オペランドで KU と指定する．もう１つの方法は，通常のマクロ形式を使って行うやり方であり，READ マクロ命令の第２オペランドはKとする．ただし，READ マクロ命令で使う DECB と WRITE マクロ命令のそれとは異ったものにする必要があるが，バッファ・アドレスだけは一致させることが要求される．したがって，動的バッファ割当てを適用している場合だと， READ 命令の DECB から WRITE 命令の DECB の所定のフィールドにバッファ・アドレスを移さねばならないことになる．ともあれ，ここでは後者の方法を例にデータ・セットのレコードの直接更新のやり方をみよう．（前者の場合だとリスト実行形式の説明が必要になるので省略する）.

図 6-55に索引順次データ・セットの直接更新の例を示している．この例は，カード上の取引レコードで索引順次データ・セットを更新している．カード・レコードは，30桁であり１桁目から10桁目にキーがあり，11桁目から30桁目に更新情報がある．この更新情報で索引順次データ・レコードの31桁目から50桁目を更新しようとしている.

DCB マクロ命令では，例によって MACRF パラメータで使用するマクロ形式を制御プログラムに知らせる．動的バッファ割当ての READ マクロ命令

```
         :
READCD   GET     INFILE, CDWORK
         READ    INDECB, K, ISDCB, 'S', 'S', KEY
         WAIT    ECB=INDECB
         TM      INDECB+24, X'FD'  DECBの例外状態コードをテスト
         BM      RDERROR           エラー分析ルーチンへブランチ
         MVC     OUTDECB+12(4), INDECB+12  バッファ・アドレス
                                           を移す.
         L       3, INDECB+16  論理レコードのアドレスをレジスター
                               3へ
         MVC     30(20, 3), UPDATE   レコードの更新
         WRITE   OUTDECB, KU, ISDCB, 'S', 'S', KEY
         WAIT    ECB=OUTDECB
         TM      OUTDECB+24, X'FF'  DECBの例外状態コードをテスト
         BM      WRERROR            エラー分析ルーチンへブランチ
         B       READCD
RDERROR  ↓

WRERROR  ↓

CDWORK   DS      0CL30
KEY      DS      CL10
UPDATE   DS      CL20
INDEX    DS      CL2000
ISDCB    DS      DSORG=IS, MACRF=(RS, WU), MSHI=INDEX,    C
                 SMSI=2000
         :
```

図 6-55　索引順次データ・セットの直接更新

(RS) とレコード更新のための WRITE マクロ命令 (WU) を使うために，MACRF=(RS, WU) とする.

データ・セットを OPEN したあと，まずカード・レコードを読みとり，そのレコード・キーを使って索引順次データ・セット上の対応レコードの直接取出しを行う．したがって，READ マクロ命令の6番目のオペランドではカード・レコードのキーのアドレスを指定する．READ マクロ命令では動的バッファ割当てを要求している．CHECK マクロ命令の代りに WAIT マクロ命令を使用して，レコードの読みとり完了をテストしているので，例外状態コード

をチェックしエラー分析をユーザー・プログラムの中で行う必要がある. 例外状態レコードは DECB の24バイト目にあるのでこれをチェックする.

READ 命令に割当てられたと同じバッファを WRITE マクロ命令でも使用する必要があるので, バッファ・アドレスを WRITE マクロ命令で使用する DECB の所定フィールドへ移す操作とレコードの更新作業を行ってから WRITE マクロ命令を実行させる. これで, レコードの直接更新が行われるのである.

以上みてきたように, 索引順次データ・セットは順次処理を行うために QISAM が, 直接処理を行うために BISAM が用意されており多様な処理が実現できるようになっている. 索引の作成や更新は, 必要に応じてアクセス方式ルーチンが自動的に行ってくれる. また, 索引の探索も必要に応じて全く自動的に行ってくれる. あるいは, あふれ域へのレコードの移動もアクセス方式ルーチンが行ってくれる. 処理プログラムでの特別な配慮は必要ないのである.

七. 直接データ・セットの構成とその処理

直接データ・セットでは, 各レコードの**制御番号** (control number) と直接アクセス・ボリューム上の記憶位置との間に対応関係を持たせている. したがって, 制御番号さえわかれば索引の探索を行わなくても必要なレコードを直接取り出すことができる. 直接データ・セットを実際に構築するさいの規則は完全にユーザーに任かされている. したがって, レコード総数, 制御番号の分布状態, レコードの形式と大きさ, 追加・削除の頻度あるいはレコードの活動度などの諸条件を勘案して, 最適な編成のデータ・セットを作りあげることができる. 最適な編成を行えば, 索引順次データ・セットよりも効率的に特定レコードの取出しが実現できる.

直接データ・セットでは, 待機アクセス技法および基本的アクセス技法のどちらを使っても, レコードの順次処理ができる. しかし, 待機アクセス技法ではレコード・キーを読み出すことはできない. データ部分だけしか取り出せない. 基本アクセス技法を使うと, 主記憶域と入出力装置間でのデータの転送単

位がレコードであるとみなされる．それゆえ，ブロック化レコード形式の場合のブロッキング，ディブロッキングは自動的に行われない．ユーザーのプログラムでブロッキング，ディブロッキングの処理を行う必要がある．このため，FないしU形式のレコード処理を行う場合には，DCB マクロ命令のパラメータである BLKSIZE の値を LRECL の値と等しくする必要がある．V形式だと，BLKSIZE の値は，LRECL の値に4バイトを加えたものに等しくする．なお，直接データ・セットにレコードを追加したり，レコードの更新を行うときには，BLKSIZE だけを指定し，LRECL は指定しない．

直接データ・セットのレコード形式はカウント／データまたはカウント／キー／データのいずれの形式も許される．ただし，後者の形式を採用する場合には，データ・セットを構成する全レコードがキーを持つ必要があり，かつF形式のレコードであることが要求される．

1.　直接データ・セットの構成

直接データ・セットを作成するとき，レコードの制御番号と直接アクセス装置のアドレスとを関係づける方法には3種類ある．制御番号をそのままアドレスとして用いる方法，対比表を用いる方法，間接アドレスによる方法である．

1)　制御番号をアドレスとして用いる方法

この方法では，制御番号をそのまま相対ディスク・アドレスとして用いる．したがって，制御番号が分かれば，それを使って直接レコードを取り出すことができる．

たとえば，0から1999までの制御番号をもったレコードからなるデータ・セットに2000レコード分のスペースを割り当てる．そのうえで，制御番号をそのまま相対ディスク・アドレスとする．したがって，**図6-56**に示しているように0503という制御番号を持ったレコードは直接アクセス装置上のこのデータ・セットに割り当てられたスペースの503番目のレコードとして，0100の番号を持ったレコードは100番目に位置づけられるレコードとして書き出す．このような編成法だと，制御番号が参照したいレコードの位置と直接関係づけられてい

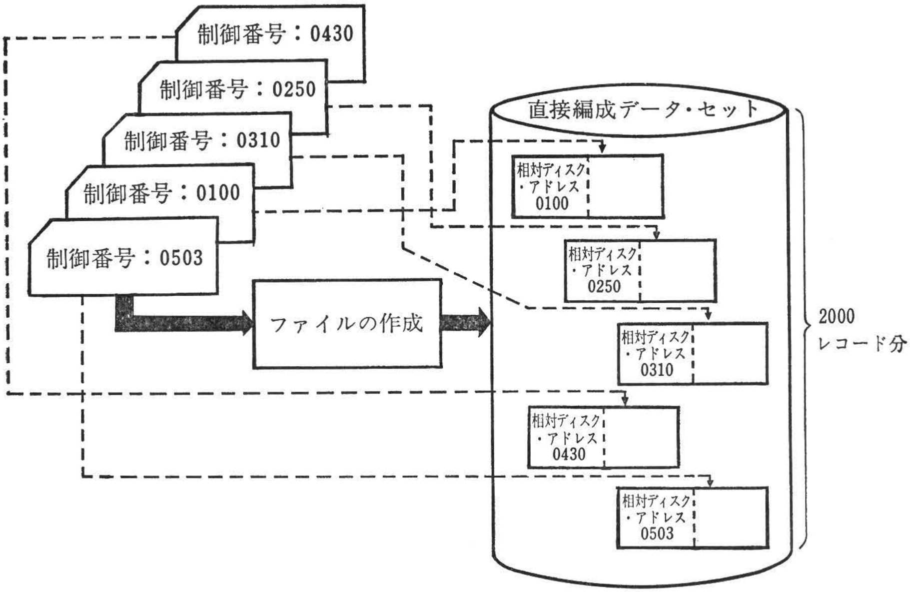


図 6-56　制御番号をアドレスとして用いる方法

るので取出しオペレーションは極めて効率的である．また，レコードは必然的に制御番号順に記憶されてしまうので，レコードの順次処理にも向いている．とはいうものの，この方法が適用できる事例はそう多くはない．というのは，制御番号は，将来の追加によってコード体系を変更しなくてすむように随所に欠番が設けられているのが普通だ．したがって，制御番号の範囲に実際に存在するレコードの数は1割にも満たないケースがしばしばである．そのために，割り当てられたスペースの大半が未使用のまま遊んでしまう．

2)　対比表を用いる方法

直接アクセス装置のスペースをレコードの制御番号ではなく，実際のレコード数にあわせて確保できれば，直接アクセス装置の記憶域がもっと有効に使える．対比表を用いるとこれが実現できる．

この方法の場合には，**図 6-57**で示したようにレコードの制御番号と直接アクセス装置上の実アドレスないしは相対アドレスとを対比したテーブルを作成する．レコードの取出しにさいしては，特定制御番号でまずこの対比表を探索し

制御番号(キー)	ディスク・アドレス
0 0 0 1 0 0	0 1 0 2 3 1 5
0 0 0 5 3 0	0 1 0 2 3 1 8
0 0 0 7 2 0	0 1 0 2 3 2 0
0 0 1 0 3 2	0 1 0 2 4 0 1
0 0 1 1 5 3	0 1 0 2 4 0 5
⋮	⋮
8 2 5 3 0 0	1 9 5 0 1 0 5
8 2 5 4 2 0	1 9 5 0 1 0 7

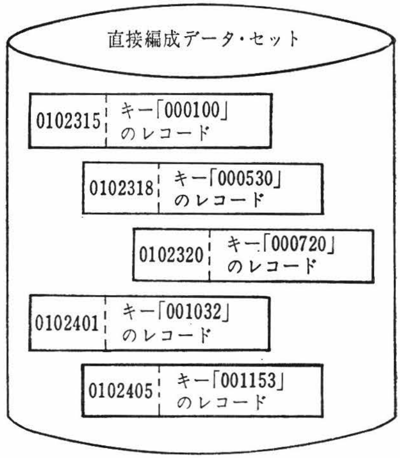


図 6-57　対比表を用いる方法

ディスク・アドレスを得てから，そのアドレスにもとづいてレコードの直接取出しを行う．したがって，この方法はレコード数が少ないとき効果的である．レコード数が多くなると，それだけテーブルのエントリーが増えるためテーブルを探索したり更新したりするのに相当の時間がかかってしまう．それゆえに，対比表を用いる方法もあまり一般的な方法とはいえない．

3）間接アドレスを用いる方法

直接データ・セットの編成手法として最も一般的な方法が，間接アドレスを用いる方法である．

間接アドレス法では，制御番号を1定の計算式を使って加工し，各レコードのアドレスにする．この操作のことを一般に**アドレス変換**（address conversion）ないしは**ランダム化**（randomizing）と呼んでいる．たとえば，**図 6-58**に示した例がそれである．この例では，除算法を用いてアドレス変換を行っている．データ・セットに割当てられたレコード・エリアの総数に一番近い素数で制御番号を割る．この余りをそのまま使えば，相対レコード・アドレスとして使える．さらに，この余りをトラック当りに収容できるレコード数で割ればディスクの実アドレスが得られるのである．



仮定：● トラック当り22レコード収容
● データ・セット全体に1000レコード収容

問題：部品番号「118165」をディスク・アドレスに変換

変換方法：

(1) 部品番号を997で割る．
118165÷997=118，余り519

(2) 余りをトラック当りのレコード数で割る．
519÷22=23，余り13
+1
14

(3) ディスク・アドレスは：
シリンダー番号 2
トラック番号 3
レコード番号 14

図 6-58　間接アドレスによる方法

アドレス変換の手法としては，除算法以外に数字分析法，重ね合わせ法，基数変換法あるいはプログラムによる自動変換などの方法がある．一般的にどの変換法がふさわしいかは言えない．それぞれのデータ・セットの性質や使用方法などの諸条件を検討して，いちばんふさわしい方法を見つけだしたり，あるいは場合によってはそのデータ・セット固有の変換法を考案すべきである．

いずれにしろ，どんな方法のアドレス変換の手法を用いても，異ったレコードの制御番号が同じアドレスに変換されてしまう可能性がある．このようなアドレスを**シノニム**（synonym）と呼んでいるが，シノニムが発生することをできるだけ少なくするような変換法を選択すべきである．また，実際のレコード数よりもレコード・エリアを多くとるとシノニムの発生は少なくなる．一般的

には，レコード・エリアの70パーセントから90パーセント程度を実際のレコードで埋めるようにデータ・セットのスペースを割当てるべきであろう．レコード単位でなく，ブロック単位あるいはトラック単位でアドレス変換するのも1つの方法である．

シノニムが生じたときの処理の仕方としては，本来のアドレスに近接する空きの場所にそのレコードを収容する方法と索引順次データ・セットのあふれ域と同じような独立した区域を設けておいて，そこにシノニムだけを収容する方法が考えられる．

最初にもふれたように，直接データ・セットを構成するやり方は，ほとんどすべてユーザーに任されている．つまり，システムが面倒みてくれる範囲は少ないのである．制御番号とアドレスの対応づけも，シノニムの処理方法もすべてユーザーが決定し，処理ルーチンを作りあげることが必要である．

2. 直接データ・セットの処理

直接データ・セットの編成法が決定すると，そのデータ・セットの取出しと更新あるいは新規レコードの追加にさいして，個々のレコードをどのような方法で参照するかを検討する必要がある．このことは，データ・セットの作成時に各々のレコードが記録された位置をフィードバックしておく必要があるかどうかを決めるためにも重要なこととなる．また，フィードバックを必要とする場合にどのような形式でフィードバックすべきかの決め手となる．オペレーティング・システムでは，次の5つの形式でレコードのアドレスを表示することができるようになっている．したがって，この5種類の中からレコードの更新，追加のさいの参照方法を採用し，必要ならデータ・セットの作成時に個々のレコードのアドレスをフィードバックさせ，その後の処理に生かすようにする必要がある．

①相対ブロック・アドレス（relative block address）：RRR

しばしば，**相対レコード番号**（relative record number）ともいう．データ・セット内のレコード（ないしはブロック）の相対的位置を3バイトの2進数で表示する．したがって，データ・セット内の最初のレコードの相対ブロック

・アドレスは X'000000' であり，２番目のレコードは X'000001' である．この形式のレコード参照方法は，固定長レコード形式に対してのみ許される．システムは，この相対ブロック・アドレスから実際のトラック番号とレコード番号を算出して，該当レコードを取出してくれるのである．

②相対トラック・アドレス (relative track address)：TTR

相対トラックを２バイトの２進数で，そのトラック上の実際のレコード番号を１バイトの２進数で表示する．したがって，最初のトラックの最初のレコードの相対トラック・アドレスは X'000000' であり，５番目のトラック上の10番目のレコードは X'000409' である．相対ブロック・アドレスとは異なり，２進数の不連続な番号となる．

③相対トラック・アドレスとキー：TT＋キー

相対トラック・アドレス（TT）とレコード・キーが収容されている主記憶域のアドレスを指定する．システムは，相対トラック・アドレスから実際のトラック・アドレスを算出する．そのあとで，該当キーを持っているレコードの探索を行う．

④相対ブロック・アドレスとキー：RRR＋キー

相対ブロック・アドレス（RRR）とレコード・キーが収容されている主記憶域のアドレスを指定する．相対ブロック・アドレスから実際のトラック・アドレスが算出され，あとはキーを使ってそのトラック上の探索が行われる．

⑤実際アドレス (actual address)：MBBCCHHR

標準の８バイト，つまり MBBCCHHR の形式で実際アドレスを指定する．この場合，Mは１バイトの２進数でデータ・エクステント・ブロック（DEB）のエントリーの相対的位置を示す．BB は２バイトの２進数でセル番号を，CC も２バイトの２進数でシリンダー番号を，HH も同じでヘッド番号を示す．BBCCHH がトラック・アドレスに対応する．Rは，１バイトの２進数で，そのトラック上のレコードの相対的順番を示す．

このような実際アドレスを使用した場合には，そのデータ・セットは**移動不可** (unmovable) の取扱いをうけるので注意が必要である．データ・セットを自由に移動することができないので，それだけ融通性に欠ける．したがって，

実際アドレスを採用するのは，特別な理由があるときだけに限定すべきである．

直接データ・セット内のレコードを参照する方法には以上の5種類があるがレコードの参照にかかわりを持つ特別な機能として，**拡張探索機能**（extended search option）とレコード更新のさいの**排他制御機能**（exclusive control option）とがある．

拡張探索機能を用いると，指定された開始点から探索をはじめ，一定トラックないしはレコード分の探索をレコードが見つかるまで継続してくれる．この機能を用いると間接アドレスを採用した場合のシノニムの処理に極めて好都合である．また，レコードの追加にさいして未使用のスペースを見つけだすのにも役立つ．なお，この機能を使用するにあたっては，DCB の中で探索トラック数ないしは探索レコード数を指示する必要がある．探索開始点には，READ または WRITE マクロ命令のオペランドで指定したアドレスが使われる．

排他制御機能は，同一ジョブ・ステップで2つ以上のタスクが同一データ・セットを同じ DCB を通じて参照するさいに用いる．つまり，同一データ・セットの中の同一レコードが異ったタスクで同時に参照されることがないように制御する目的で使われる．このことは，レコードの更新作業を行っているときに重要な意味をもってくる．目下更新中のレコードが，他のタスクによって取り出され処理されて再びそのレコードが書きもどされると折角の更新レコードが壊されてしまうからである．これを防ぐために排他制御を要求すると，WRITE または RELEX マクロ命令で排他制御を解放するまで他のタスクからそのレコードをアクセスできなくしてしまうのである．

さて，直接データ・セットの処理であるが，直接データ・セットの作成には BSAM を使用する．直接データ・セットのレコードの取出し，追加および更新には BDAM の能機を用いる．なお，直接データ・セットのレコードを記録されている順番，つまり物理的順次処理を行なうのには QSAM ないしは BS-AM を適用することができる．ここでは，BSAM による直接データ・セットの作成と BDAM による直接データ・セットのレコードの取出し，追加および更新について説明する．

1）BSAM による直接データ・セットの作成

直接データ・セットは，次の３つのいずれかの方法で作成することができる：

- レコードをあらかじめアドレスの順番に分類しておき，そのあとで物理的順番で連続的にレコードを書きだして行く．この場合，欠番のアドレス部分に対応するレコードとして見かけレコードを書き出しておく．見かけレコードの形式は**図 6-59**に示した通りであり，見かけレコード・コードおよびブロックの順序番号などは後述する特別な WRITE マクロ命令で書き出してくれる．いずれにしろ，この方法を採用した場合には，キーつきのＦ形式レコードでないといけない．
- 直接データ・セットに割当てられた全スペースにあらかじめ見かけレコードを書いておき，実際のレコードは BDAM を使って見かけレコードを更新する方法ないしはレコードを追加する方法で直接データ・セットを作りあげてゆく．この場合の見かけレコードとは，第１の方法でいう見かけレコードではない．ユーザーが適当に設定した見かけレコードのことである．したがって，レコードはキーつきであっても，キーなしであってもよい．
- 第３の方法は，可定長ないしは不定長のレコード形式の直接データ・セット

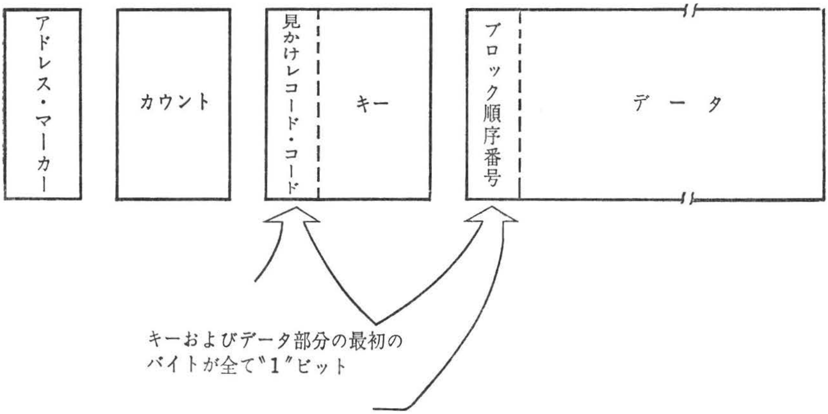


図 6-59　見かけレコードの形式

を作成するのに好都合な方法である．つまり，BSAM を使って各トラックの最初に**トラック記述レコード**（R0 ないしは capacity record ともいう．詳しくは，第5章の「DASD のハードウェア特性」を参照のこと）だけを書き出しておき，あとで BDAM を使って空きのスペースにレコードを追加するやり方で直接データ・セットを作成してゆく．

第1の方法，第2の見かけレコードの書き出しおよび第3のトラック記述レコードの書き出しのいづれの場合も，順次データ・セットの作成処理とほとんど同じやり方をする．したがって，DCB マクロ命令のデータ・セット編成

〔記号名〕	WRITE	DECB名，タイプ，DCBアドレス，バッファ・アドレス[, 長さ / , 'S'] 〔, 次のアドレス〕

●DECB 名……DECB に割当てる記号名．

●タイプ ……WRITE マクロ命令のオペレーション・タイプ．つぎのいずれかのコードを記入する．

- ●SF ……新規データ・ブロックがデータ・セットに追加される形式で書き出される．
- ●SFR …次のアドレスがフィードバックされるという点を除いて SF と同じ．
- ●SD……キーつきの固定長ブロックに出してのみ適用され，データ・セットに見かけブロックを書き出す．
- ●SZ ……可変長または不定長ブロックに出して適用され，トラック記述レコード（R0）を書き出す．

●DCBアドレス……DCB のアドレスを指定．

●バッファ・アドレス……データ・セットに追加するブロックを貯えている主記憶域のアドレスを指定．キーを使用する場合には，このエリアのデータの先頭につけておく必要がある．
SZ の場合には，このアドレスは無視されるので省略してよい．
SD の場合には，キーの長さに1バイト追加した大きさでよい．制御プログラムが見かけレコードを作り出して出力する．

●長さ……不定長ブロックに対してのみ指定．出力すべきデータのバイト数．'S' を指定すると，DCB からブロックの長さがとられる．

●次のアドレス……つぎのレコードが書き出される相対トラック・アドレスをフィードバックするフィールドのアドレス．

表 6-9　直接データ・セット作成用の WRITE マクロ命令の一般形式

フィールドでは**物理的順次**（physical sequential），つまり DSORG=PS または PSU を指示する．しかし，DD ステートメンドでは直接アクセス（DSORG=DA または DAU）を指定しなければならない．さらに，DCB マクロ命令では，直接アクセス装置であることを指定する必要がある（DEVD=DA）．キーつきのレコードである場合には，キーの長さ（KEYLEN）も指示する必要がある．なお，直接データ・セットを作りあげるための WRITE マクロ命令を使用する旨の指示を MACRF=(WL) で行う．WL は，BDAM のための直接データ・セットを作りあげる WRITE マクロ命令を使用することを意味する．この直接データ・セットを作りあげるための特別な WRITE マクロ命令の形式は，**表 6-9** に示した通りである．

図 6-60 で第 1 の方法による直接データ・セットの作成例を示している．この例では，あらかじめキーの順序に分類された 204 桁のレコードをもとに直接データ・セットを作成している．各レコードが持っている 4 バイトの 2 進数字のキーは10進数で1000から8999にわたっており，8000レコード分のスペースが要求されている．

DD ステートメントでは，DSNAME 以外に DCB パラメータで DSORG=DA の指定が必要である．BLKSIZE, KEYLEN, RECFM パラメータは DCB マクロ命令で指定してもよい．また，SPACE パラメータで，この直接データ・セットに要するスペースを要求する必要がある．

DCB マクロ命令の DSORG=PS, MACRF=(WL) および DEVD=DA の指定に関しては先述した通りである．

入力レコードから直接データ・セットのレコードを作り出す場合には，WRITE マクロ命令の第 2 オペランドで SF と指定し，第 4 のオペランドでレコードの入っている主記憶域のアドレスを指示する．一方，入力レコードに欠番があるものに関しては見かけレコードを書き出す．この場合の WRITE マク命ロ令では，第 2 オペランドで SD を，第 4 オペランドではキーの長さに 1 バトを加えた大きさの見かけレコード書き出し用のエリアを指定する．これによって，レコードの欠けているアドレスには見かけレコードだけを書き出してくれるのである．

DDステートメント

```
// OUTDD    DD      DSNAME=DIRECT. MAST5, DCB=(DSORG=DA,  C
//                  BLKSIZE=200, KEYLEN=4, RECFM=F),      C
//                  SPACE=(204, 8000), ……
// TAPEIN   DD      ……
```

処理プログラム

```
                :
          L       9, =F'1000'    制御番号を1000にセット
          OPEN    (DALOAD, (OUTPUT), TAPEDCB)
          LA      10, COMPAPE    COMPAREのアドレスをレジスター
                                 10へ
NEXT      GET     TAPEDCB
          LR      2, 1
COMPARE   C       9, 0(2)     入力レコードのキーと制御番号を比較
          BNE     DUMMY
          WRITE   DECB1, SF, DALOAD, (2)    データ・レコードの書き
                                            出し
          CHECK   DECB1
          A       9, =F'1'     制御番号を1増やす.
          B       NEXT
DUMMY     C       9, =F'8999'  8000レコードすべてが書き出されたか？
          BH      ENDJOB
          WRITE   DECB2, SD, DALOAD, DUMAREA   見かけレコードの
                                               書き出し
          CHECK   DECB2
          A       9, =F '1'    制御番号を1増やす
          BR      10    LASTルーチンを実行するまではCOMPAREへブ
                        ランチ
LAST      LA      10, DUMMY   DUMMYのアドレスをレジスター10へ
          BR      10          DUMMYへブランチ
ENDJOB    CLOSE   (TAPEDCB,, DALOAD)
                :
DUMAREA   DS      CL5     見かけレコード書出し用エリア
DALOAD    DCB     DSORG=PS, MACRF=(WL), DDNAME=OUTDD,    C
                  DEVD=DA, ……
TAPEDCB   DCB     EODAD=LAST, DDNAME=TAPEIN,             C
                  MACRF=(GL), ……
                :
```

図6-60　直接データ・セットの作成

図 6-61では，見かけレコードを書き出すことによって直接データ・セットを作成するための前準備をしている．キーをもたない200バイトのレコードを1000個分書き込めるスペースを確保し，各レコードのカウント部とデータ部とを書き出すことによって，データ・セットのエリアの形式を整えている．データ部はすべて2進数の0として書き出している．この場合の WRITE マクロ命令の第2オペランドも SF ないしは SFR である．なお，スペース全体の書き込みが終了したかどうかはレジスター15に返される**戻りコード** (return code) をテストすればよい．このようにしてあらかじめ形式を整えた直接データ・セットに対して，後述する BDAM の WRITE マクロ命令を使って所定の個所にレコードを挿入することによって直接データ・セットを完成すればよいのである．

DDステートメント

```
// NEW      DD      DSNAME=DAFILE, DCB=(DSORG=DA, BLKSIZE  C
//                  =200, RECFM=F), SPACE=(200, 1000), ……
```

処理プログラム

```
                :
            OPEN     (DADCB, (OUTPUT))
MORE        WRITE    DADECB, SF, DADCB, OUTDUM
            C        15, =F'8'  戻りコードのテスト（スペース一ぱいにかい
                                たか？）
            BNL      OUT
            CHECK    DADECB
            B        MORE
OUT         CLOSE    (DADCB)
                :
            DS       0D
OUTDUM      DC       200X'00'
DADCB       DCB      DSORG=PS, MACRF=(WL), DDNAME=NEW,
                :
```

図 6-61 直接データ・セット作成のための前準備 (preformat)

```
          :
          OPEN     (NEWFIL, (OUTPUT))
          LA       9, 1000
AGAIN     WRITE    DECB1, SZ, NEWFIL
          CHECK    DECB1
          BCT      9, AGAIN     1000トラック書いたか?
          CLOSE    (NEWFIL)
NEWFIL    DCB      DSORG=PS, MACRF=(WL), DEVD=DA,            C
                   DDNAME=OUT, ……
          :
```

図 6-62 直接データ・セット作成のための前準備（トラック記述レコードの書き出し）

第3の方法であるトラック記述レコードを各トラックに書く場合の例を示したのが図6-62である．この例では，可変長レコードの直接データ・セットを作成するための準備として1000トラックに記述レコードを書き出している．このための WRITE マクロ命令では，第2オペランドで SZ を指定し，第4番目のオペランドは何も指定しない．このようにして準備された直接データ・セットには，後述する WRITE マクロ命令で各トラックにレコードを追加してゆく方法でレコードを書き出し，データ・セットを完成してゆく．

2) BDAM による直接データ・セットのレコードの取出し

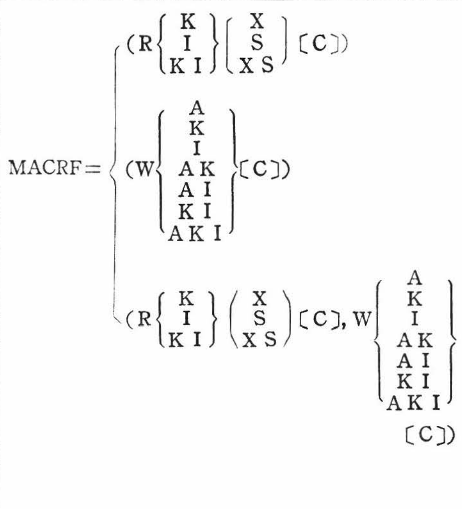
BDAM で使われる DCB マクロ命令は，他の場合の DCB マクロ命令とほとんど変らない．指定できるパラメータの数は他の場合よりも少ないくらいである． BDAM で使用される DCB マクロ命令の主要なパラメータをまとめたのが表6-10である．BDAM にはブロッキング，デブロッキングの機能はない．したがって，BLKSIZE パラメータはあっても LRECL パラメータはない．当然のことながら，ブロッキング，ディブロッキングが必要ならユーザー・プログラムで行う．BDAM 特有のパラメータとしては LIMCT がある．先述した**拡張探索機能**（extended search option）を使用する場合の探索範囲をこれで指定する．相対ブロック・アドレスを用いてレコードを参照しているときだけ探索するのに必要なブロック数を指定し，他の場合には探索トラック数を指定

する. また, この機能を用いる場合には, OPTCD=E を指示する必要があ

タパラメータ	指　定　方　法
BLKSIZE=	入出力ブロックのバイト数を指定.
DDNAME=	データ定義名を指定.
DSORG=	DA(Direct Access) または DAU (Direct Access Unmovable) を指定. ただし, 直接データ・セットの作成のときは PS または PSU を指定.
KEYLEN=	キーの長さをバイト数で指定. ただし, すでにデータ・セットが存在する場合にはこの情報はデータ・セット・ラベルから得られるので指定しなくてよい.
LIMCT=	拡張探索機能 (OPTCD=E) が要求されている場合の探索範囲を指定. 相対ブロック・アドレスが使われている場合には探索ブロック数を指定. その他の場合には, 探索トラック数を指定.
OPTCD={W, E, F, {R, A}}	A : READ/WRITE マクロ命令で, 装置の実際アドレスを使用することを要求. E : 拡張探索機能を使用することを要求. F : READ/WRITE マクロ命令でフィードバック機能を使用することを要求. F を省略するとブロックの実アドレスがフィードバックされる. R : READ/WRITE マクロ命令で相対ブロック・アドレスを使用することを要求. W : 書き出しオペレーションで妥当性チェック (validity check) を要求. なお, A も R も指定されていないと, 相対トラック・アドレスが要求されているものとみなされる.
RECFM={U, V, F, FT}	使用するレコード形式を指定. U : 不定長レコード V : 可変長レコード F : 固定長レコード T : 必要なら, レコードをあふれトラックに書き出す.
MACRF=	使用するマクロ形式を指定 (別表を参照のこと)
SYNAD=	入出力エラー分析ルーチンのアドレスを指定.

表 6-10　BDAM の DCB マクロ命令で使われる主要なパラメータ

る．OPTCD ではこの他レコードを参照するさいの方法（実際アドレス，相対ブロック・アドレスまたは相対トラック・アドレスのいずれか）およびアドレスのフィードバックの機能を用いるかどうかを指示する．なお，レコードを参照するさいにキーを用いるかどうかは，READ/WRITE マクロ命令のオペランドで指定する．また，BDAM の MACRF パラメータでもこのことを指示しなければならない．DBAM の MACRF パラメータの指定は，若干複雑なので表6-11に別にまとめてあるので参照されたい．



A：データ・セットにブロックを追加する場合.
C：CHECK マクロ命令を使用する場合（省略すると WAIT マクロ命令を使用することを意味する）.
I：ブロック標識で探索する場合.
K：キーで探索する場合.
R：READ マクロ命令を使用する場合.
S：動的バッファ割当てを要求する場合.
W：WRITE マクロ命令を使用する場合.
X：排他制御を要求する場合.

表 6-11　BDAM の MACRF パラメータの指定方法

さて，直接データ・セットのレコードの直接取出しであるが，やはりREAD マクロ命令を使う．順次取出しの場合には，QSAM の GET マクロ命令か BSAM の READ マクロ命令が使える．BDAM の READ マクロ命令の一般形式は表6-12に示した通りである．読み取りオペレーションのタイプは他の場合より増えている．ブロック標識を使った探索とは，キーを用いないで実際アドレス，相対ブロック・アドレスないしは相対トラック・アドレスだけで該当レコードを探し出す方法である．一方，キーを使った探索とは，相対ブロック・アドレスないしは相対トラック・アドレスをもとに探索すべきトラックを決め，そのあとで該当レコードを見い出すのにキーを使って探索する方法であ

+0	ECB (1バイト)	例外状態コード (2バイト)	
+4	オペレーションのタイプ (2バイト)	ブロックの長さ (2バイト)	
+8	DCB のアドレス (4バイト)		
+12	データ・エリアのアドレス (4バイト)		
+16	ブロック参照アドレス (4バイト)		
+20	キー・アドレス		

図 6-63　BDAM の DECB の形式

る．後者の場合には必ずキーつきのレコード形式であることが要求される．また，排他制御を行うかどうかも，このオペレーション・タイプのコードで指示する．

取り出すべきブロックの相対トラック・アドレス，相対ブロック・アドレスあるいは実際アドレスの入っている場所のアドレスは READ マクロ命令の7番目のオペランドで指定する．キーを使った探索を行う場合には6番目のオペランドで必ずキー・アドレスも指定し，取出したいキーをあらかじめこの場所に入れていく必要がある．

BDAM の場合に READ/WRITE マクロ命令で作り出される DECB の形式は BISAM のそれとは若干異る．図 6-63でその形式を示しておいたので必要のつど照参されたい．

さて，直接データ・セットのレコードの取出しの具体的例をみよう．図6-64図6-65および図6-66がそれである．いずれの場合にも，サンプルを簡単にするために，レコードを参照するアドレスは定数としてあらかじめ確保している．実際には，このアドレスは他の入力媒体から得られたキーを加工して得るものである．

まず，図 6-64の例からみよう，これは実際アドレスを用いてレコードの探索を行う例である．したがって，DCB マクロ命令では OPTCD=A として実際アドレスを使用することを要求する．さらにMACRF パラメータで，READ マクロ命令ではブロック標識を使った探索を行う旨（RI）を指示する．MACRF の SC は動的バッファ割当てと CHECK マクロ命令を要求するため

のものである．なお，実際アドレスを使用すると**移動不可**（unmovable）とな

〔記号名〕	READ	DECB名，タイプ，DCBアドレス，{バッファ・アドレス / 'S'}，{長さ / 'S'}，{キー・アドレス / 'S'}，ブロック・アドレス〔，次のアドレス〕

- DECB名……DECBに割当てる記号名．
- タイプ……読みとりオペレーションのタイプ．
  - DI………ブロック標識を使ったブロックの探索．データとキー（もしあれば）の両方が取り出される．
  - DK………キーを使ったブロックの探索．データだけが取り出される．
  - DIF……フィードバック機能が働く点を除いてDIと同じ．
  - DIX……排他制御とフィードバック機能が働く点を除いてDIと同じ．
  - DKF……フィードバック機能が働く点を除いてDKと同じ．
  - DKX……排他制御とフィードバック機能が働く点を除いてDKと同じ．
  - R………上記タイプのサフィックスとして使用．「次のアドレス」オペランドの指定を次のレコード・アドレスとみなす．
  - RU………上記タイプのサフィックスとして使用．「次のアドレス」オペランドの指定を次のレコード・アドレスないしは装置アドレスとみなす．
- DCBアドレス……読みとりのためのDCBアドレス．
- バッファ・アドレス……ブロックを読みこむ主記憶域のアドレス．動的バッファ割当ての場合は 'S'．この場合，割当てられたアドレスはDECBにおかれる．
- 長さ……読みとりデータのバイト数．'S' を指定するとバイト数はDCBから得られる．
- キー・アドレス……取り出したいレコードのキー・エリアのアドレス．キーによる探索の場合にはあらかじめキーをこの場所に入れておく．そうでない場合にはキーが入れられる．バッファ・アドレスの個所で 'S' としている場合には，ここも 'S'．キーを取り出さない場合は0を指定，
- ブロック・アドレス……取出すべきブロックの相対トラック・アドレス，相対ブロック・アドレスないしは実際アドレスの入っている主記憶域のアドレス．もし，フィードバック機能が要求されていると，ブロックの装置アドレスがここに入れられる．
- 次のアドレス……次の順番にあるレコードのアドレスが入れられる主記憶域のアドレス．この指定を使う場合には，「長さ」オペランドで 'S' を指定のこと．

表 6-12　BDAM の READ マクロ命令の一般形式

```
                :
             READ      DECB1, DI, INDCB, 'S', 'S', 0, RECADDR
             CHECK     DECB1
               |       取出しレコードの処理 (レコードのアドレスは DECB1
               ↓       +12にある)
RECADDR      DS        0D
EXTENT       DC        X'01'        M
BIN          DC        AL2(0)       BB
CYL          DC        AL2(150)     CC
TRACK        DC        AL2(5)       HH
RECORD       DC        X'06'        R
                :
INDCB        DCB       DSORG=DAU, MACRF=(RISC), OPTCD=A, ……
                :
```

図 6-64 直接データ・セットのレコードの取出し (実際アドレスによる方法)

るので DSORG=DAU とする.

READ マクロ命令では, ブロック標識を使ってレコードの探索を行なうために第2オペランドで DI と指定する. 特にキーを取り出す必要がない場合には6番目のオペランドは0とする. 7番目のオペランドで8バイトの実際アドレスが入っている場所を指示する. 実際アドレスは, 必ずダブルワードから始まる場所に2進数で入れておく必要がある. このマクロ命令が実行されると実際アドレスで指示された場所のレコードを主記憶装置に読み込む. 読み込まれたレコードが収容された主記憶域のアドレスは DECB の12バイト目から始ま

```
                :
             READ      DECB2, DK, INDCB, 'S', 'S', KEYADDR, RECADDR
             WAIT      ECB=DECB2
               |
               ↓       例外状態コードのチェックと取出しレコードの処理
KEYADDR      DC        CL5'1234A'
RECADDR      DC        AL2(511)     TT
                :
INDCB        DCB       DSORG=DA, MACRF=(RKS), ……
                :
```

図 6-65 直接データ・セットのレコードの取出し (相対トラック・アドレスとキーによる方法)

る4バイトに入れられているので，このアドレスを基点にして，レコードの各フィールドを処理すればよい．

図6-65では，相対トラック・アドレスとキーを使ってレコードを取り出している．DCB マクロ命令の OPTCD で何らのパラメータも指定しないと，相対トラック・アドレスを用いることを暗示する．MACRF では，READ マクロ命令でキーによるブロック探索を行う旨（RK）指定する．READ マクロ命令ではキーによる探索を行なうので第2オペランドで DK とする．第6オペランドではキーの入っているエリアを，第7オペランドでは相対トラック．アドレスが入っている2バイトの場所を示す必要がある．相対トラック・アドレスも2進数表示である．なお，図6-65では WAIT マクロ命令を使用しているので，このマクロ命令を実行したあと DECB の2バイト目から始まる2バイトの例外状態コードの内容をチェックしたあとで，取り出したレコードの処理をする必要がある．

図6-66では，相対ブロック・アドレスを用いてレコードの取出しを行っている．データ部分だけでなくキーの部分も取り出している例である．相対ブロック・アドレスを使用することから，DCB マクロ命令の OPTCD ではRを指示する．MACRF は RISC である．これは，図6-64の場合と同じである．READ

```
                 :
                READ     DECB3, DI, INDCB, 'S', 'S', KEYADDR, RECADDR
                CHECK    DECB3
                  ↓      取出しレコードの処理
                FREEBUF DECB3, D, INDCB
                 :
KEYADDR        DS       CL6
RECADDR        DC       AL3(249)
                 :
INDCB          DCB      DSORG=DA, MACRF=(RISC), OPTCD=R,             C
                        KEYLEN=6, ……
                 :
```

図6-66　直接データ・セットのレコードの取出し（相対ブロック・アドレスによる方法）

マクロ命令の第2オペランドではDIを指定する．取り出されるキーを収容する場所を6番目のオペランドで，取り出すべき相対ブロック番号が入っている場所は第7オペランドで指定する．相対ブロック・アドレスは，あらかじめ3バイトの2進数でこの場所に入れておく必要がある．

3) BDAMによる直接データ・セットへのレコードの追加とレコードの更新

直接データ・セットへのレコードの追加方法は，レコードの形式と使用されている編成法に大きく依存する．したがって，まずこの点に関して簡単に言及しておこう．

①キーつきのF形式レコードの場合

この形式の場合のデータ・セットの作成は，「BSAMによる直接データ・セットの作成」の項で説明した1番目の方法で行われるのが普通である．したがって，追加の処理は，見かけレコードの更新によって達成される．この場合のレコードの参照は，相対ブロック・アドレスであっても相対トラック・アドレスであってもよい．

相対ブロック・アドレスを使ってレコードを追加する場合だと，システムがそのアドレスを相対トラックに変換してくれる．そのあとで，そのトラックを探索し，トラック上の最初の見かけレコードの場所に追加レコードを書き出す．もしも，そのトラック上に見かけレコードが存在しない場合にはその旨知らせてくれる．しかし，拡張探索機能を要求しておけば，指定された範囲内の最初の見かけレコードの場所に新規レコードを書き出す．万一，その範囲内にも見かけレコードがない場合には，書き出しが行えなかったことを処理プログラムに知らせてくれる．いずれにしろ，見かけレコードが存在するなら自動的に追加を行ってくれるのである．

相対トラック・アドレスを用いてレコードを参照する場合にも同じ方法がとられる．該当トラック上の最初の見かけレコードないしは拡張探索機能を要求しているとその範囲内の最初の見かけレコードの場所にレコードを追加する．

②キーなしのF形式レコードの場合

この形式の場合だと，データ・セットの作成はあらかじめ全スペースに見か

けレコードを書いておいて，これにレコードを追加する方法で行う．つまり，「BSAM によるデータ・セットの作成」の項で説明した2番目の方法である．したがって，データ・セットの新規作成もレコードの追加も全く同じ方法で行われるのである．この方法の場合も，レコードの追加は，実際にはデータ・セットにすでに存在している見かけレコードを更新することで達成される．ただし，キーつきのF形式の場合のように自動的に追加は行われない．というのは，データ・セット作成の前準備で作りあげられた見かけレコード自体が自動的に書かれたものでないからである．見かけレコードの標識はプログラマーが独自に定めた方法にしたがっている．それゆえに，レコードを追加するにさいしては，まず READ マクロ命令でレコードを取り出して見かけレコードかどうか標識を調べ，見かけレコードの場合には，更新の方法で追加レコードを書き出す．したがって，WRITE マクロ命令としては，追加の形式（MACRF＝WA）ではなく更新の形式（MACRF＝W）を使用する必要がある．

③キーつきのVまたはU形式レコードの場合

この場合のレコードを追加する手法は，データ・セットの編成法に依存する．間接アドレスの方法を用いているか対比表を用いているかである．

間接アドレスを用いてデータ・セットを作成する場合だと，あらかじめ各トラックは初期化しておかねばならない．つまり，「BSAM による直接データ・セットの作成」の項で説明したように各トラック記述レコードを書いておく．したがって，トラック上の使用可能なスペースはトラック記述レコードに絶えず表示されている．この情報を使って，追加レコードが該当トラック上に収容しきれるかどうか検査される．

対比表を用いる場合だと，レコードはスペースの最初から空きのないようにつめて入れられてる．したがって，レコードを追加するに十分な後続トラックの初期化を行ってから追加処理をする必要がある．

いずれの場合にも，新規レコードを追加するには相対トラック・アドレスしか用いられない．システムは，トラック記述レコードを参照して，該当トラック上に十分な空きのスペースがあるかどうか確める．追加レコードを収容するに足るスペースがあると，新規レコードが書き出される．拡張探索機能を用い

ると，探索範囲内の最初の空きスペースにレコードが記録されるのは言うまでもない．

④キーなしのVまたはU形式の場合

この形式では，レコードを追加することはむずかしい．相対トラック・アドレスか実際アドレスで既存のデータ・セットからレコードを取り出すだけである．

さて，レコードの追加であるが，そのためには WRITE マクロ命令を用いる．**表6-13**に示したのが BDAM の WRITE マクロ命令の一般形式である．オペランドの指定は， READ マクロ命令の場合とほとんど同じである．この WRITE マクロ命令を用いてレコードを追加している例が**図6-67**である．この例は，さきの図6-60で作成したデータ・セットにレコードを追加している．し

```
                :
         OPEN     (DIRECT, (OUTPUT), TAPEIN)
NEXT     GET      TAPEIN, TPWORK
         L        4, KEY
         SH       4, =H'1000'      相対ブロック番号の準備
         ST       4, REF
         WRITE    DECB, DA, DIRECT, DATA, 'S', KEY, REF+1
         WAIT     ECB=DECB
         CLC      DECB+1(2), =X'0000'     例外状態コードのチェック
         BE       NEXT
           ↓      入出力エラーの分析ルーチン
TPWORK   DS       0F
KEY      DS       F
DATA     DS       CL200
REF      DS       F
                :
DIRECT   DCB      DDNAME=ADDD, DSORG=DA, RECFM=F,                C
                  KEYLEN=4, BLKSIZE=200, MACRF=(WA),             C
                  OPTCD=R, ……
TAPEIN   DCB      DSORG=PS, MACRF=(GM), ……
                :
```

図6-67 直接アクセス．データ・セットへのレコードの追加

〔記号名〕　WRITE　DECB名，タイプ，DCBアドレス，{バッファ・アドレス / 'S'}，{長さ / 'S'}，{キー・アドレス / 'S'}，ブロック・アドレス

- DECB名……DECBに割当てる記号名.
- タイプ………書き出しオペレーションのタイプ.
  - DA ……新規ブロックの追加.「ブロック・アドレス」オペランドで指示したアドレスから空きスペースの探索を行う.
  - DAF……書き出したブロックの装置アドレスが「ブロック・アドレス」オペランドで指示した場所にもどされる点を除いてDAと同じ.
  - DI ……「ブロック・アドレス」オペランドで指示したアドレスにブロックを書き出す.
  - DIF ……ブロックの装置アドレスが「ブロック・アドレス」オペランドで指示した場所にもどされる点を除いてDIと同じ.
  - DIX……排他制御が解除される点を除いてDIFと同じ.
  - DK ……探索引数として「キー・アドレス」オペランドで指示したキーを使ってブロックを書き出す. ブロックの探索は,「ブロック・アドレス」オペランドで指示したアドレスから開始される.
  - DKF……ブロックの装置アドレスが「ブロック・アドレス」オペランドで指定した場所にフィードバックされる点を除いてDKと同じ.
  - DKX……排他制御が解放される点を除いてDKFと同じ.
- DCBアドレス……書き出しオペレーションのDCBアドレス.
- バッファ・アドレス……ブロックが入っている主記憶域のアドレス. 動的バッファ割当てで得られたエリアにブロックが入っているときだけ 'S'.
- 長さ……出力すべきブロックのバイト数. 'S' を指定するとバイト数はDCBから得られる.
- キー・アドレス……キーが収容されている主記憶域のアドレス. 動的バッファ割当てで得られた記憶域にブロックが入る場合には 'S'. キーを出力しないときは0.
- ブロック・アドレス……出力オペレーションで使用する相対ブロック・アドレス. 相対トラック・アドレス, 実際アドレスを収容する記憶域のアドレス.

表 6-13　BDAM の WRITE マクロ命令の一般形式

たがって，キーつきのF形式レコードの場合であり，キーとデータ・レコードの両方を書き出している．指定されたトラックに見かけレコードが存在しない場合にはその旨の表示がDECBの例外状態コードのフィールドに戻される．

これをテストするには CHECK マクロ命令ではなく，WAIT マクロ命令を用いる．見かけレコードがない場合には，独立あふれ域を使ってもよいし，次のトラックの見かけレコードを見つけてそこにレコードを挿入してもよい．すべてその方法は，ユーザーに委ねられている．

DCB マクロ命令では，WRITE マクロ命令を使ってレコードを追加することを指示する（MACRF＝WA）．また，キーつきのレコードであることから KEYLEN パラメータも指定する．相対ブロック番号を用いて書き出しを行うので OPTCD＝R とする．WRITE マクロ命令を与えるまえに，あらかじめ相対ブロック番号を準備し，このマクロ命令の第7オペランドで指示している場所に入れておく．キーつきのレコードを書き出すので，第6オペランドで指示した場所にはキーを入れておく．WRITE マクロ命令を実行したあとで，WAIT マクロ命令を与え DECB の例外状態コードをチェックすれば，追加レコードが正しく書き出されたかどうか判明する．

最後に直接データ・セットの更新の例を示そう．**図6-68**がそれである．データ・セットは5バイトのキーをもったレコードから構成されている．キーは00001から25,000までの範囲にあり，各データ・レコードは100バイトである．キーを相対ブロック・アドレスに変換してレコードを参照している．データ・レコードの最初30バイトを更新して，直接データ・セットに書きもどす．

DCB マクロ命令では，相対ブロック・アドレスでレコードを参照するので，OPTCD＝R とし，ブロック標識でレコードを探索して更新するためにマクロ形式は MACRF＝(RISC, WIC) とする．更新のための入力レコードを読んだあと，入力されたキーを相対ブロック・アドレスにするために2進数に変換して，1を引く．相対アドレスは，1ではなくゼロから始まるので1だけ引いたのである．READ マクロ命令では，ブロック標識による探索を行うので，第2オペランドは DI とする．キーを読み込む必要はないので第6オペランドは0とする．読み込んだレコードのアドレスは DECB の12番目から始まるフィールドにあるので，これを基点にレコードを更新する．更新が終ったら，出力レコードのアドレスを WRITE マクロ命令の DECB の所定の場所に移してから，この命令を実行すればよい．これが，直接データ・セットのレコードの

```
                :
               OPEN    (DIRECT, (UPDAT), TPDCB)
NEXT           GET     TPDCB, TPWORK
               PACK    KEYFIELD, KEY
               CVB     3, KEYFIELD      キーを2進数に変換
               SH      3, =H'1'         相対ブロック番号にするため1引く.
               ST      3, REF
               READ    DECBRD, DI, DIRECT, 'S', 'S', 0, REF+1
               CHECK   DECBRD
               L       3, DECBRD+12     レコードの記憶域アドレスをレジスタ
                                        ー3へ
               MVC     0(30, 3), DATA   レコードの更新
               ST      3, DECBWR+12
               WRITE   DECBWR, DI, DIRECT, 'S', 'S', 0, REF+1    更新書出し
               CHECK   DECBWR
               B       NEXT
                :
KEYFIELD       DS      0D
               DC      XL3'0'
TPWORK         DS      0CL35
KEY            DS      CL5
DATA           DS      CL30
REF            DS      F
DIRECT         DCB     DSORG=DA, DDNAME=UPDD, MACRF=                C
                       (RISC, WIC), OPTCD=R, BUFNO=1, BUFL=100
TPDCB          DCB     DSORG=PS, MACRF=(GM), DDNAME=TAPIN, …
                :
```

図 6-68　直接データ・セットのレコード更新

更新である.

以上のように，直接データ・セットの作成は BSAM を用いて行い，そのデータ・セットのレコードの取出し，追加および更新を BDAM で行うことができるのである.

第7章

ジョブ管理の諸機能とその構造

一. 資源管理の歴史的発展過程

コンピューティング・システムを導入する最大の目的は，それを使って問題解決の仕事を行うことにある．この目的を経済的かつ効率的に遂行するためには，導入したシステムを最大限にこき使う必要がある．可能なかぎりシステムを多忙にしておくことである．1分たりとも遊ばせてしまうことは無駄を意味する．この目的達成のためにオペレーティング・システムが出現した．コンピューティング・システムの生産性の向上に寄与することこそが，オペレーティング・システムの根源的目標であった．この目標を共通基盤にして，スループットの向上，ターンアラウンド・タイムの短縮あるいはプログラマー，オペレータ，管理者などブレーンウェアへ各種の便宜を提供することを派生的目標としている．これらの目標に対処するためには，データ処理システムの資源，つまりハードウェア資源，情報資源および人的資源（図7-1参照）を効果的に活用しなければならない．

ところで，初期段階のデータ処理システムの資源は比較的容易に管理できた．要員の数は少なかったし，コンピューティング・システムも小型で，使用する入出力装置の種類や数も少なかったからだ．しかも，コンピューティング・システムで処理する仕事は簡単でその数は限られていた．しかし，データ処理システムの規模が大きくなり，システムが複雑・高度化するにつれて，コンピューティング・システムそれ自体を，システム資源を管理する道具として活用

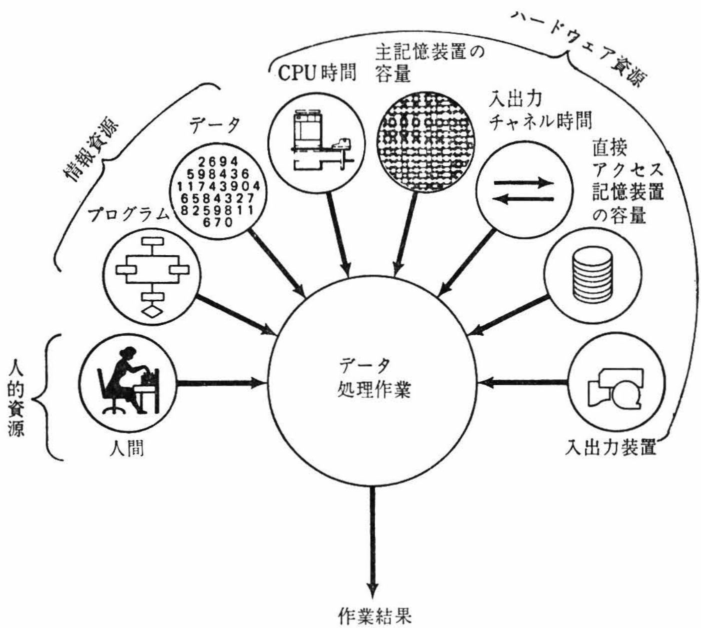


図 7-1　主要なシステム資源

しはじめた．この萌芽的な代表的例は，すべてのプログラムを保存管理するために補助記憶装置を用いたことであった．

一定形式の制御カードを読み取り，補助記憶装置から特定プログラムを捜し出してくる特殊なプログラムが作られた．このプログラムが初期の**制御プログラム**である．特定プログラムをみつけ出すと，制御プログラムはその問題処理プログラムを主記憶装置にロードした．そのあとで，処理プログラムを実行するために制御権を渡す方法をとった．この制御プログラムは一般には主記憶装置に常駐したので，呼び込んでくる問題処理プログラムを格納するに十分なスペースを主記憶装置に残しておく必要があった（図 7-2 の上側参照）．

システム資源を管理するうえでの共通的なサービス機能がどんどん制御プログラムに組み入れられ，制御プログラムはその機能を拡充していった．しかし，システム資源の有効利用をソフトウェア面だけでカバーするのには限界が

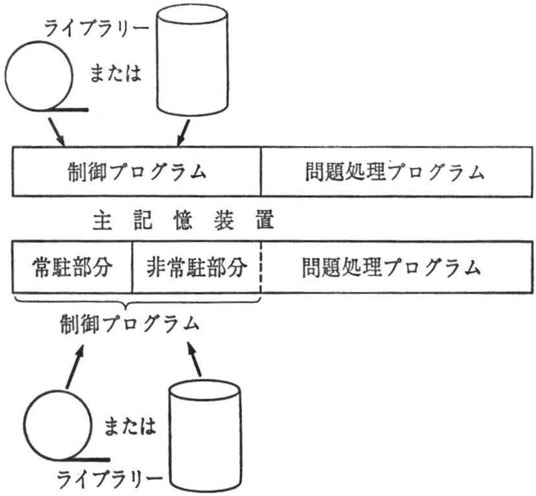


図 7-2　主記憶装置の利用の仕方

あった．その結果，制御プログラムをその機能面で援助してゆくための工夫がハードウェアの設計思想に反映され，制御プログラムの概念を拡大していった．種々の特別な内部回路が設計され，組み込まれた．割込み機能，システム状態の切換え機能，記憶保護機能あるいはプログラムの再配置機能などが制御プログラムをささえるハードウェア機能として出現した．このようにして，制御プログラムはますます強力になっていった．しかし，制御プログラムが複雑になればなるだけ，そのプログラムを常駐させておくに必要な主記憶装置はどんどん増大することになった．その結果，問題処理プログラムに残されるスペースは少なくなってゆく．

当然のことながら，主記憶装置という資源をもっと有効に利用する方法が追求されることになった．これを克服するための１つの方法が，制御プログラムを論理的なセグメントに分割し，使用頻度の少ないセグメントは補助記憶装置上に記憶しておくことであった．ライブラリーが設けられた装置は，オペレーション中は何時でも制御プログラムでアクセスできる状態にしておく．したがって，ライブラリー上にある制御プログラム・ルーチンが必要になると，直ちに主記憶装置に呼び込み，特別なサービス機能を果させる．この種の制御プロ

グラム部分を**非常駐ルーチン**（nonresident routine）という．制御プログラムは，常に主記憶装置に**常駐している**（resident）部分と，必要のつど補助記憶装置上のライブラリーから呼び出され一時的に主記憶装置に留まる非常駐部分から構成されるようになった．これら制御プログラム部分で占められる記憶域以外が問題処理プログラム用の記憶域となる（**図7-2**の下側参照）．

制御プログラムの非常駐部分が主記憶装置を占めるやり方には2通りの方法がある．1つの方法は，図7-2の下側に示しているように非常駐部分が入る特定域を別に設けておく．この場所に必要のつどオーバレイする．別の方法は，処理プログラムで使用される区域に非常駐部分を呼び込む方法である．この場合には，処理プログラムと非常駐制御プログラム部分が互いに排他的でないと都合が悪い．一方が不用になったとき，他方がそれで占められていた記憶域にオーバレイされる必要があるからだ．したがって，**ジョブ・スケジューラー**（job scheduler）のように，処理プログラムと処理プログラムとの間に介在するような制御プログラムにしかこの方法は適用できない．それゆえ，2つの方法をたくみに組み合せて非常駐制御プログラムを主記憶域に呼び込む方法がとられる．

常駐（resident）と**非常駐**（nonresident）という工夫の唯一の欠点は，非常駐部分が必要になったとき，それを捜し出し主記憶域に呼び込むのに一定の時間を要することにある．それゆえ，制御プログラムで占める記憶域を節約する代価として，制御プログラムの実行速度を落してしまうことになる．したがって，制御プログラムの設計にあたっては，主記憶装置の効率的利用と実行時間の短縮という相反する要求を上手にバランスさせることが必要となる．制御プログラム作成時に工夫が必要なだけでなく，その利用にさいしてユーザーのニーズに合わせた選択が効くような配慮も必要だ．そのために，制御プログラムのどの範囲までを常駐にするかの選択を，オペレーティング・システムの生成時に行えるようにしている．

システム資源の有効管理に関して，別のもっと重要な面にも目が向けられた．コンピューティング・システムの内部速度が向上するにつれて，ほとんどのプログラムがCPU資源の大半を無駄にしていることがはっきりしてきたからだ．CPUの利用率が10パーセントから20パーセント程度のプログラムが

かなりあることが分った．その結果，資源全体をより効率的に活用するために，2つ以上の処理プログラムの実行を併行して行わせ，それらのプログラムにうまく資源を分配するような工夫がなされた．これが，**多重プログラミング**(multi-programming) である．多重プログラミングは，言い換えると**多重ジョブ処理** (multi-job processing) のことであり，多量の仕事と沢山の資源を持ち合わせている場合にふさわしい方法である．多重ジョブ処理とは，2つ以上の仕事を同時に処理することに他ならない．

多重プログラミングは，別の問題を提供した．確かに CPU 資源の有効利用に貢献するが，他のハードウェア資源に一考を要することになった．DASDのようなハードウェア資源は，複数の処理プログラムで同時に共用することが可能だ．しかし，磁気テープ装置，カード読取装置，カード穿孔装置あるいは印刷装置などのハードウェア資源は一度に1つの処理プログラムでしか利用できない．それゆえに，制御プログラムがある時間帯に4つのジョブを開始させようとし，しかもそれぞれのジョブが共用できないハードウェア資源の利用を求めている場合だと，そのうちの唯一つのジョブにしかその資源を使わせることができない．別のジョブはそのジョブが装置の使用を終ったあとにはじめて開始できる．これでは，多重プログラミングは実現できない．1つの解決策は，この種の共用不可能な装置を複数台設置する方法である．しかし，これは安易なやり方だ．システム資源の有効利用という出発点をあやうくしてしまう．CPU資源は効率的に利用できても，入出力装置を沢山増やしたのでは個々の装置の効果的利用を約束しないからだ．

当然のことながら，別の方法が追求された．その結果，この問題解決に奉仕するための特別なプログラムが工夫された．特に，カード読取・穿孔装置と印刷装置を有効利用するために考え出された．つまり，カード読取装置からの入力データを読み取って，いったんすべてを共用可能な装置，つまり DASD 上に書き出してしまう特別なプログラム (reader program : **読取プログラム**) とDASD上にはき出された出力データを印刷装置やカード穿孔装置に出力する別なプログラム (output writer : **出力書出しプログラム**) とである．ユーザーのプログラムでカード読取装置からデータを読み取ったり，カード穿孔装置や印

刷装置に直接データを書き出すことはしない．その代り，これらの特別なプログラムで共用可能な装置への読み書きを行う．ユーザー・プログラムの実行に先立って，カード読取装置からのデータを読取プログラムが事前に読み込んで DASD 上に書き出してしまう．ユーザー・プログラムでは，あたかもカード読取装置からデータを読み取っているかのごとく DASD 上からカード・データを入力し処理する．穿孔データや印刷データは，あたかも穿孔ないしは印刷している積りで DASD 上に一時的に貯えられる．ユーザー・プログラムの実行が終了すると，出力書出しプログラムが自動的に機能する．すなわち，DASD 上に出力されているデータを読み取って，カード穿孔装置ないしは印刷装置に書き出す．これらの処理は，他の問題処理プログラムの実行と完全に併行して行われる．このように，カード読取・穿孔装置や印刷装置は特別なプログラムの入出力装置として割当て，多くのユーザー・プログラムには割当てない方法をとる．これが多重プログラミングにおける共用不可能な装置に関する問題を解決する極手となった．この方法は，単に多重プログラミング上の問題を解決しただけに留まらない．通常の処理プログラムの中で，カード読取装置や印刷装置を用いると，その特性からプログラムの実行時間は著しく低下してしまう．そればかりではない．カード読取装置や印刷装置も処理のための待ちが生じて本来の性能速度で稼動することができないのが実情である．ところが，前述の方法を採用すると，処理プログラムの実行速度ははやまるし，入出力装置は本来の性能に近い速度で稼動する．その結果，全体としての処理効率は一段と高まり，システム資源の有効利用に貢献するのである．なお，このような方法を一般に**スプール**（SPOOL : Simultaneous Peripheral Operation On-Line または spooling）と呼んでいる（**図7-3**参照）．

さらに，多重プログラミングの技法が進んでくると**多重タスキング**（multi-tasking）の概念が導入されるようになった．この概念が導入されることによって，システム資源の有効利用は一段と深まってゆく．タスクとは，一般に達成すべき仕事を意味する．コンピューティング・システムでの達成すべき仕事とは，CPU でデータを処理することである．データ処理の仕事を遂行するために，CPU は一連の命令を実行する．プログラム，サブプログラム，ルーチン

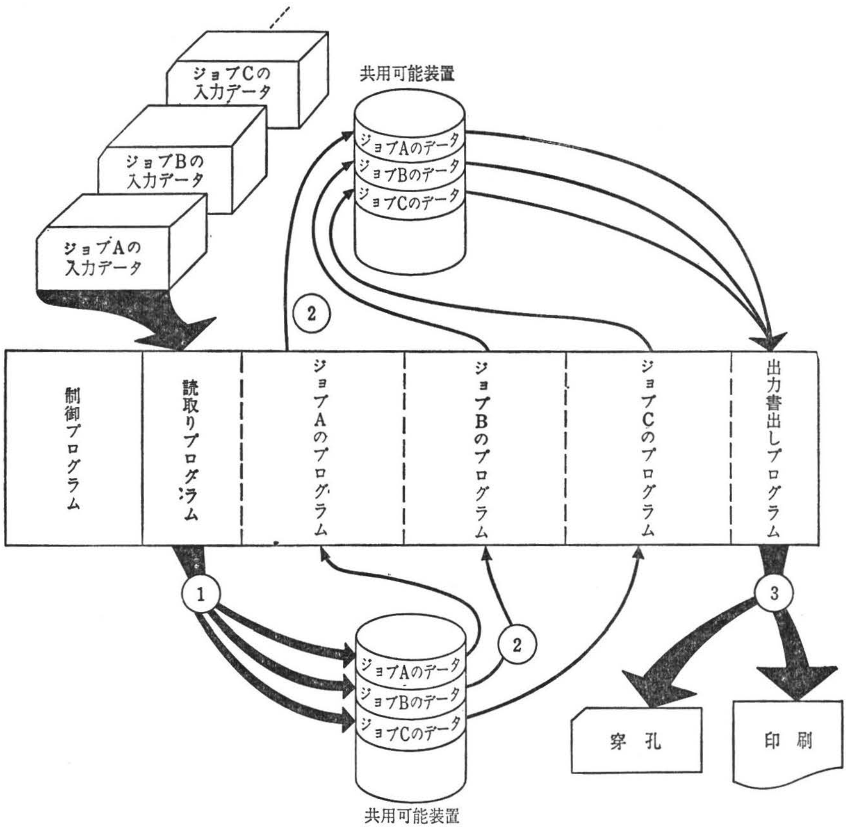


図 7-3　スプールの基本構造

あるいはサブールチンなど様々な呼び方がなされているものは，いずれも一連の命令群からなる．つまり，データ処理上のタスクとは，プログラムないしはプログラム部分を実行することによって CPU で遂行される仕事である．そのために，入出力装置やデータあるいは主記憶装置などの資源を利用する．タスクとは，システム資源を競合して利用することのできる独立した仕事の単位といえるのである．したがって，このタスクにシステムの諸資源を巧妙に割りふることによって，システム全体の生産性を向上させるような仕組みを制御プログラムに組み込む方法が目論まれてきたのである．

これまで，主として主記憶装置を中心に資源管理の発展過程をふりかえってみた．一方では，ジョブを連続的に処理してゆくことによって，コンピューティング・システムの遊休時間を排除することにチャレンジしてきた．コンピューティング・システムの処理速度が向上し，データ処理の仕事量がふえるにつれて，オペレータとコンピューティング・システムとの間での差異がますます明白になり，見逃すことができなくなってきた．オペレータが定型的な仕事を行っている間，あるいは次に何を行うべきかを思案している間，コンピューティング・システムが無駄に待っている時間が非常に多くなってきた．問題は，オペレータがデータ処理の機械的な部分にあまりにも介入し過ぎるという単純な事実であった．オペレータがどんなに迅速に，また確実に仕事を行ったとしても，しょせん限界がある．コンピューティング・システムの処理速度と信頼性が増せば増すだけ深刻な問題となってきた．コンピューティング・システムに仕事を行わせるための**準備をする時間**（setup time）の方が，コンピューティング・システムが仕事を行うために稼動している時間よりも多くかかることがしばしば生じてしまう結果になった．事実，オペレータがテープ・リールをセットしたり，カード読取装置に穿孔カードを置いたり，操作卓のオペレーションに費やしている時間のために，1日のうちの半分以上の時間を無為に過ごしてしまうコンピューティング・システムがかなりみられたのである．小さい仕事を数多くこなす計算室の場合には，オペレータの介入によって無駄に費やされる時間は特に深刻な問題であった（図7-4右側参照）．

コンピューティング・システムの処理速度はどんどん向上してゆく．しかし，オペレータの手作業による準備時間は，いかに熟練したとしてもその短縮には限度がある．したがって，コンピューティング・システムでの仕事はより速く処理できても，仕事と仕事の間でのオペレータが行う準備の時間はほとんど変りがない．つまり，コンピューティング・システムの処理能力の向上はこの種の無駄な時間を埋め合わせることに何ら貢献をしなかった．そのため，コンピューティング・システムの処理速度が向上すればするだけ，オペレータ待ちの時間の比率は大きくなってゆく結果となった．

この種の遊休時間を排除するために，いくつもの独立した仕事をできるだけ

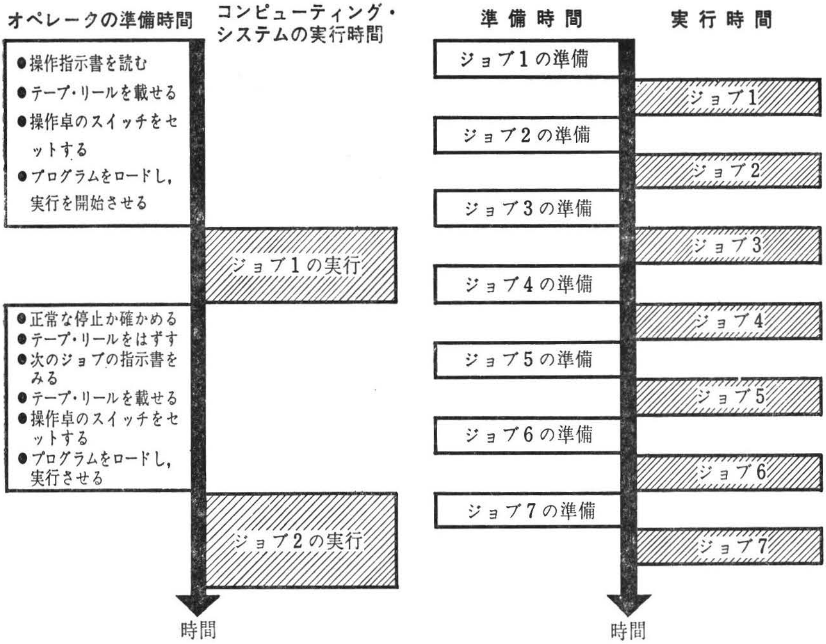


図 7-4 ジョブとジョブの間のコンピューティング・システムの遊休時間

オペレータの介入を少なくして，連続的に実行できるようにコントロールする工夫がなされたのは当然の帰結であった．過去にオペレータが行ってきた機能の大部分を代行する制御プログラムを作ったことで，その目的は十分達成された．ジョブ処理の機械的な部分に人間が関係する度合いを制御プログラムが減らしてくれたおかげで，ジョブはより速くかつより効果的に処理できるようになり，人間の誤りに影響される率も少なくなった．またそのおかげで，オペレータはシステムの全般的な操作を計画したり指示したりすることに，より多くの時間がさけるようになった．

制御プログラムは，その目的を達成するために，まずプログラマーとコンピューティング・システムとの仲介者という不合理な役割からオペレータを解放しなければならなかった．そのために**ジョブ制御言語** (Job Control Language :

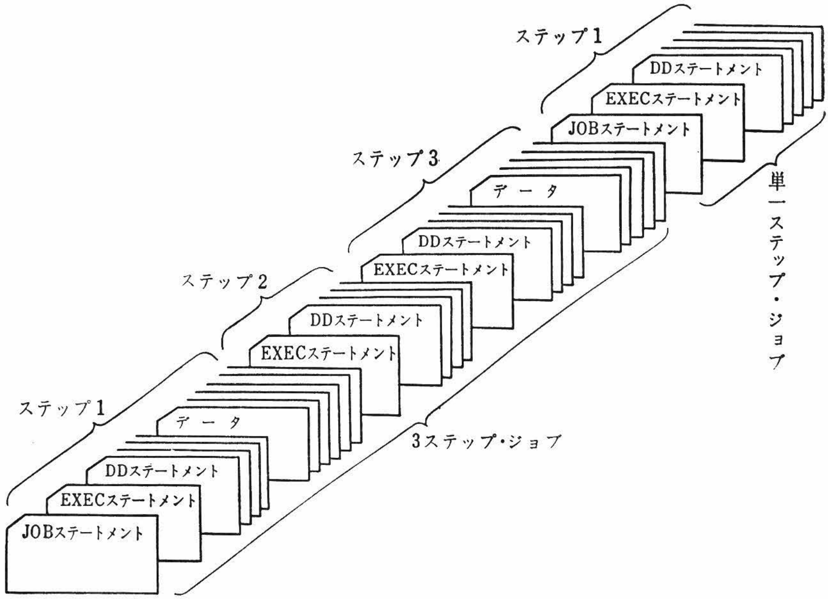


図 7-5　ジョブの定義

略して JCL）を設けて，オペレータを経由せずにプログラマーが行いたいと思う仕事（**ジョブ**および**ジョブ・ステップ**）の正確な定義を制御プログラムに直接伝えるようにした．この言語は，いくつかの**ジョブ制御ステートメント**（job control statement）から成り立っていて，穿孔カードに記録され，そのあと読み取られて解読され，それに従って制御プログラムが働らくようにしたものであった．その中の，JOB ステートメントはジョブの始まりを識別表示し，ジョブを指定するのに使用される．また，EXEC ステートメントはジョブ・ステップの始まりを指定し，そのジョブ・ステップを実行するための特定プログラムを名前で指示するのに使用される．ほとんどのオペレーティング・システムでは，各ジョブに１つ以上のジョブ・ステップを含めることができるようになっている（図 7-5 参照）．

JOB ステートメントと EXEC ステートメント（または，これに相当するス

テートメント）は，JCL の中でも特別重要なもので，よく使用されるものである．大部分の制御プログラムは，その他にもいくつかのステートメントを読み，解読し，それに応答するように設計された．その正確な数や種類は制御プログラムによって異っている．しかし，それぞれのジョブ・ステップで使用されるファイルないしは入出力装置に関しての指示を行うステートメントはいずれの制御プログラムも用意されていた．図 7-5 で示した DD ステートメントがこれに相当する．また，オペレータに対するメッセージを制御プログラムに中継するような指示を行うステートメントなども用意された．

コンピューティング・システムの遊休時間を減らすために，1つのジョブまたはジョブ・ステップから別のジョブまたはジョブ・ステップへの切替えを自動的に制御するように制御プログラムが設計された．そのために，ジョブの定義を行うための特別な制御言語が用意された．そして，制御プログラムでジョブを連続的に処理してゆくためには，いくつものジョブ定義を順々に並べて，ジョブ定義の束をあらかじめ用意しておくようにした（図7-5参照）．そのあとこのジョブ定義の束を特定の入力装置に入れるようにした．ジョブ定義はジョブ入力装置に連続的に並べられているので，1つのジョブまたはジョブ・ステップが終了するとすぐに制御プログラムは次のジョブを読んで，その処理を始めることができるのである．このようにして，オペレータの介入を最少限にして，ジョブの連続した流れを読み，処理することができるようになった．この手法が，**バッチ・ジョブ処理**（batched job processing）または**連続式ジョブ処理**（stacked job processing）と呼ばれているものである．

システム資源の有効利用には2つの側面がある．その1つは，システムの稼動中に遊びが生じている資源を出来るだけ少なくし，個々の資源が本来的にもっている性能を可能なかぎりひき出すことである．主記憶装置の効果的利用だとか，多量プログラミングによる CPU の効率的利用あるいはスプールの構想などはこれに当る．もう1つの側面は，コンピューティング・システムが仕事の準備や故障のために遊休してしまう時間を最少限にくいとめることである．このためには，前述した仕事の連続処理や RAS の機能などがある．さて，本章では，ジョブを連続的に処理しオペレータの負担を出来るだけ少なくするこ

とを主眼とした制御プログラムの機能について詳述することにしよう．これらの機能は総称して**ジョブ管理**（job management）**の機能**と呼ばれることは第1章ですでにふれた通りである．

二．ジョブ管理の機能とその基本構造

1.　ジョブ管理の主要な機能

ジョブ管理の主要な目的は，コンピューティング・システムを可能なかぎり作動している状態におくことにある．オペレータの介入によって，システムが遊休状態におちいることを最大限排除する．そのためには実行すべき仕事がシステムで処理できる状態になったとき，それを，即座に処理可能な態勢にしておく必要がある．ジョブ管理のための制御プログラムは，1つあるいはそれ以上のジョブがコンピューティング・システムで連続的に処理されてゆくように指示し，かつコントロールを行うことによって，上記目的を達成する．

ジョブ（job）とは，オペレーティング・システムが行う作業の最も大きい単位であり，他のジョブとは独立している．ジョブは，1つ以上のジョブ・ステップから構成される．ジョブ・ステップは，1つの処理プログラムが関係する仕事の単位，つまり，その処理プログラムで遂行される仕事のことである．換言すると，ジョブは総合的な処理業務の単位であり，毎月の給与計算とか毎日の商取引業務処理といった業務に関連する1つ以上の処理プログラムで遂行される仕事の集まりといえる．このような各ジョブは，プログラマーないしはオペレータがコーディングした一連のジョブ制御ステートメントによって定義されることは前にもふれた．

ジョブの定義は，そのジョブに関する情報（たとえば，ジョブ名やジョブの優先順位）を指定する JOB ステートメントで与える．各ジョブは，1つ以上のジョブ・ステップで成り立つので，それらのステップを定義し，実行する順番に並べておく必要がある．このためには，EXEC ステートメントを用いる．このステートメントで，そのステップを実行するための処理プログラムの名前や

それを実行するのに必要な主記憶装置の大きさなどを指示する．

このようにして，個々のジョブ定義を次々に並べたジョブの連続を作ることができる．このようなジョブの連続のことを一般に**入力ジョブの流れ**（input job stream）と言う．この入力ジョブの流れを制御プログラムで読み取らせ，ジョブまたはジョブ・ステップの間でコンピューティング・システムを停止させずに，連続的に処理することができる．あるジョブに対してオペレータの措置（テープ・リールを載せるなど）が必要なら，その旨のメッセージを出して指示する．またはあらかじめ空きの装置に次のジョブで必要とするリールをセットしておけばよい．なお，入力ジョブの流れはカード読取装置から入力しても，磁気テープ装置から入力してもよい．場合によっては，**遠隔ジョブ入力システム**（remote job entry system）のように電信回線を通して入力することもある（図7-6参照）．

該当ジョブ・ステップの中で処理されるデータ・セットに関しては，すべてDDステートメントで定義する必要がある．制御プログラムは，DDステートメントで定義された情報にもとづいて入出力資源の割り当てを行ってくれる．特別な場合には，入力データを入力ジョブの流れの中に置くこともできる（図7-6参照）．

以上のような入力ジョブの流れにもとづいて，ジョブ管理用の制御プログラムが機能してくれるのである．その主要な機能は，次の6つにまとめうる：

① **入力ジョブの流れの読取りと分析**：入力ジョブの流れを読み取り，ジョブ制御ステートメントと他のものを見分ける．制御ステートメントの場合だと，その内容を解釈し分析する．そのあとで，システムに対して各ジョブを記述するのに必要な制御用のテーブルを準備する．

② **入出力装置の割当て**：各ジョブ・ステップごとに，必要とする入出力装置がすべて割り振られたかどうか確かめる．直接アクセス記憶装置の場合だと，要求通りのスペースが割り当てられたかどうか確認する．さらに，オペレータが必要な磁気テープおよび直接アクセス・ボリュームを取り付けたかどうかを確かめる．

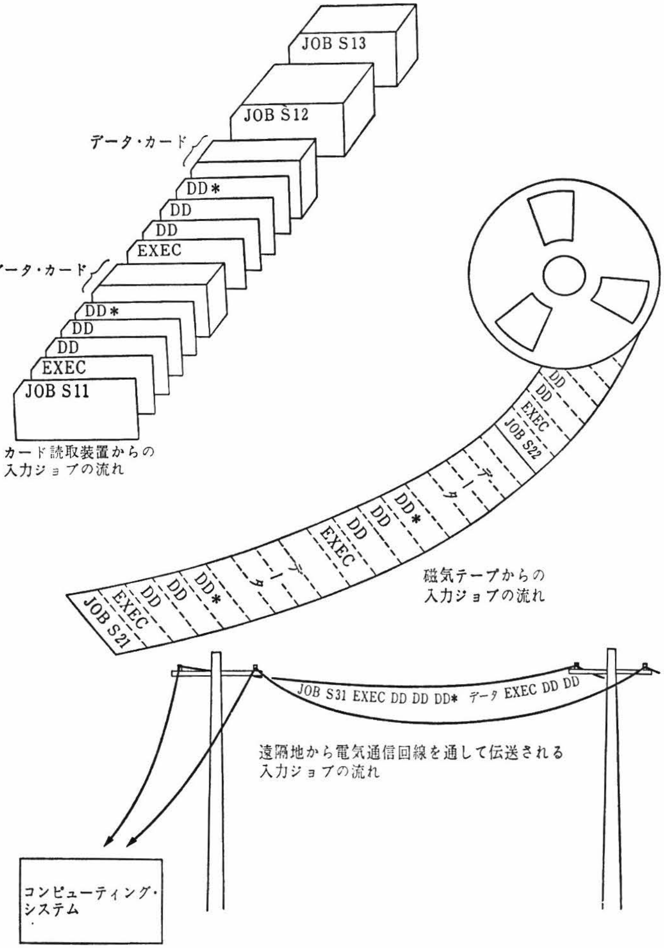


図 7-6　入力ジョブの流れの例

③　**ジョブの全体的なスケジューリング**：実行すべきジョブを先着順あるいは優先順位にもとづいて選択し，実行できる状態にする．

④　**入力データの変換**：入力ジョブの流れにあるデータを見分け，そのデータを直接アクセス記憶装置上に一時的に記憶させる．

⑤　**出力データの変換**：処理プログラムで直接アクセス記憶装置上に一時的に書き出された出力データを，他のプログラムの実行と併行して，印

刷装置やカード穿孔装置上に出力する．

⑥ **オペレータとシステム間の連絡**：システム操作卓からオペレータが入力した**指令**（command）を解読し，しかるべき処理を行う．あるいは，逆にオペレータに対してシステムからの指示をメッセージの形式で出力する．

以上のような機能を果すためのジョブ管理プログラムは，**マスター・スケジューラー**（master scheduler）と呼ばれるプログラム部分と**ジョブ・スケジューラー**（job scheduler）と呼ばれるプログラム部分から構成される．

マスター・スケジューラーは，オペレータとシステムとのコミュニケーションの仲介役を果す．また，ジョブ・スケジューラーで遂行される仕事のスケジュールや開始を行うためにオペレータが使用することもある．

一方，ジョブ・スケジューラーは，入力ジョブの流れを読み取り，解読し，一連のジョブの実行をスケジュールし，ジョブ・ステップの実行を開始させる．また，ジョブやジョブ・ステップの実行後の後始末の面倒も引きうける，ジョブ・スケジューラーには，2種類のタイプがある．**順次式ジョブ・スケジューラー**（sequential job scheduler）と**優先式ジョブ・スケジューラー**（priority job scheduler）である．いずれのタイプのジョブ・スケジューラであっても，つぎのような3つの部分から成り立っている：

① 読取／解釈プログラム（Reader/Interpreter：略して R/I）
② 開始／終止プログラム（Initiator/Terminator：略して I/T）
③ 出力書出しプログラム（Output Writer または単に Writer）

2. ジョブ・スケジューラーの機能構造とジョブ制御言語

前述したようにジョブ・スケジューラーは，実行すべき仕事の準備を行い，順次式あるいは優先式順序でジョブのスケジュールを行う．ジョブ・スケジューラーは，ジョブ制御言語で定義されたジョブないしはジョブ・ステップに関する情報を読み取って解読し，ジョブ・ステップの実行を開始させる．ジョブ・ステップの実行が終るとその後始末をしたうえで，次のジョブ・ステップに

関して同様のことを行う．また，ジョブ・ステップの実行中に DASD 上に出力されたデータを出力書出しプログラムを用いて，印刷装置や穿孔装置上に最終結果としてはき出す．つまり，スプールの機能も果す．

1）ジョブ・スケジューラーの機能構造

ジョブ・スケジューラーの機能構造を図式化したのが**図 7-7** である．この図を参照しながらジョブ・スケジューラーの機能構造の概略について説明しておこう．

① 読取／解釈プログラム

読取／解釈プログラムには，入力ジョブの流れを読み取る責任が負わされている．入力ジョブの流れのレコードを読み取り，選別を行う．ジョブ制御ステ

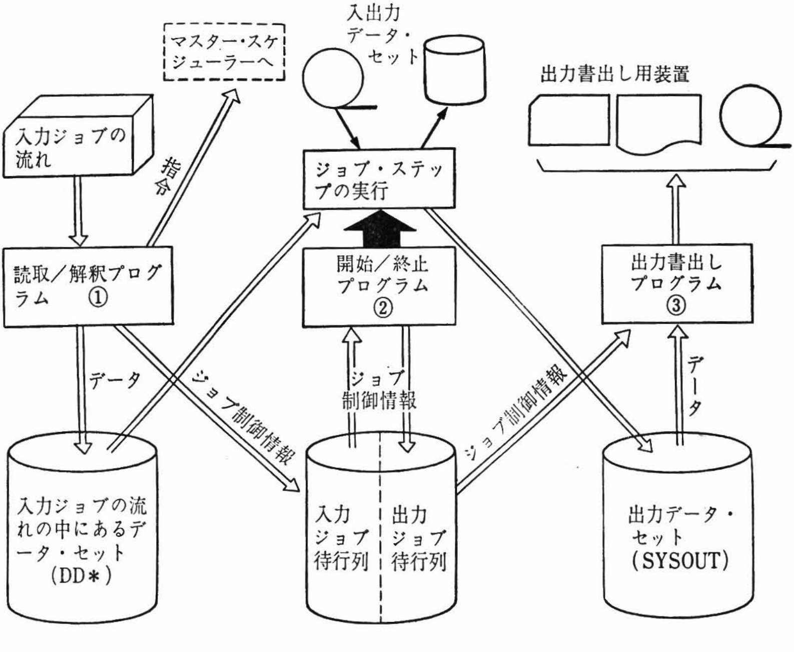


図 7-7　ジョブ・スケジューラーの機能構造（優先式ジョブ・スケジューラーの場合）

ートメントだと，その内容を分析する．そのあとで，ジョブ・ステップを開始し実行するのに必要となる情報を一定形式の制御テーブルに集約する．入力ジョブの流れの中に**指令**（command）が入っている場合だと，その情報をマスター・スケジューラーへ引き渡す．優先式ジョブ・スケジューラーの場合だと，ジョブ制御情報を優先順に**入力ジョブ待行列**（input job queue または input work queue）に移す．一方，入力ジョブの流れにあるデータは直接アクセス装置上に移しかえる．

② 開始／終止プログラム

開始／終止プログラムは，入力ジョブの流れないしは入力ジョブ待行列のいずれかから，次に実行すべきジョブ・ステップを選択する．そのあとで，そのジョブ・ステップで必要となる入出力装置を調べ，その割り振りを行う．必要なら，オペレータにボリュームの取り付け指示を出し，そのボリュームが指示された装置にセットされたかどうか確認する．ただし，この方法だとオペレータが取り付けを行うのに時間を要し，無駄が生じてしまう．そこで，オペレータが取り付けに要する浪費時間を最少限にするために**自動ボリューム認識**（Automatic Volume Recognition：略して AVR）の機能を用いることができるようになっている．

自動ボリューム認識機能を用いる場合には，空いている磁気テープ装置や磁気ディスク装置上にラベルのついたボリュームをあらかじめ取りつけておく．ジョブ・スケジューラーは，各ボリュームの識別データとそのボリュームがセットされた装置の記録をとっておく．ジョブの準備のために特定ボリュームが必要になると，まずその記録テーブルを探索する．テーブルを調べた結果，必要とするボリュームがすでに取り付けられていることが分かれば，ボリューム取り付け指示のメッセージを出す必要はなくなる．この結果，オペレータがボリュームをセットするために処理の進行が一時さまたげられるという事態を回避することができる．この機能は，一般には作業計画が事前に立てられ，かつくり返し行われるような環境のもとで有効に活用することができる．このような場合には，オペレータはどのボリュームがどのような順序で用いられるかを

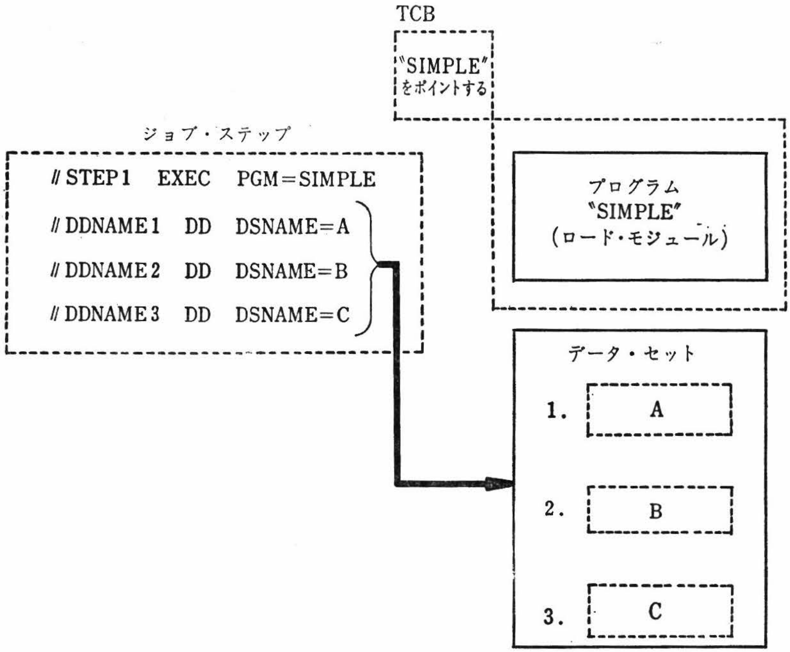


図 7-8　ジョブ・ステップとタスクの相互関係

事前に判断することができるからである．したがって，この自動ボリューム認識機能は順次式ジョブ・スケジューラーにしか適用不可能である．優先式ジョブ・スケジューラーの場合だと，ジョブの優先順位にもとづいてシステムが自動的に次に実行すべきジョブを決定してしまうからである．

話しをもとにもどそう．開始／終止プログラムは，実行すべきジョブないしはジョブ・ステップを選択し，必要な入出力装置と DASD 上のスペースの割当てを行う．そのあとで，このステップを実行するに必要なタスタを生成する(図7-8参照)．ジョブ・ステップの実行にさいしては，入力ジョブの流れの中にあったデータも入力データとして取り出され，処理される．また，出力データを出力書出しプログラムで印刷装置やカード穿孔装置に出力する予定だと，それらデータは DASD 上に出力データ・セットとして一時的に書き出される．

ジョブ・ステップの実行が終了すると，再び開始／終止プログラムが機能する．入出力データ・セットに関して，ジョブ制御ステートメントで指示された後始末を行う．さらに，優先式ジョブ・スケジューラーの場合には，出力データ・セットに関する制御情報を**出力ジョブ待行列**（output job queue または output work queue）に貯える．これは，あとで出力書出しプログラムが出力ジョブを選択するのに必要な情報である．

③ 出力書出しプログラム

優先式スケジューリング・システムのもとでタスクが実行されているときには，出力データセットを一時的に DASD 上に記憶させておくことができる．そして，後で出力書出しプログラムを使って，そのデータを**システム出力装置**（system output device：通常，印刷装置またはカード穿孔装置）に出力する．このシステム出力装置は，出力書出しプログラム用タスクによって制御される．さらに，出力装置は使用別クラスにグループ化することもできる．たとえば，ある一台の印刷装置は高い優先順位をもち，かつデータ量の少ない印刷出力用のクラスに指定する．一方，別の1台の印刷装置はデータ量の多い印刷出力用のクラスに指定するといった具合いである．なお，出力データ・セットをどのクラスの出力装置に割り当てるかは DD ステートメントで指定する．

出力ジョブ待行列も優先順位にもとづいて管理されるようになっているので，出力書出しプログラムも当然のことながら優先順位にしたがって出力ジョブ待行列内のジョブを選択し，処理を行うのである．

2） ジョブ制御言語の働き

ジョブ・スケジューラーが，ジョブの連続処理を遂行するのに必要となる情報は，ジョブ制御言語を使って与える．ジョブ・スケジューラーとシステム・ユーザーとの間のコミュニケーションは，つぎの6種類の制御ステートメントで行う：

① ジョブ・ステートメント
② 実行ステートメント

③　データ定義ステートメント

④　指令ステートメント

⑤　区切りステートメント

⑥　終了ステートメント

これらのステートメントのオペランドに書かれるパラメーターでは，ジョブ・スケジュラーがジョブおよびジョブ・ステップの実行を規制したり，データの取り出しや事後処理を行ったり，入出力資源を割り当てたり，またオペレータと連絡したりするための情報を指示する．以下で各ステートメントの機能と用法を要約しておく．これらステートメントの具体的使用法は本章の後半で詳述する予定である．

①　ジョブ・ステートメント（JOB ステートメント）

このステートメントは，ジョブの始まりを示す．入力ジョブの流れに2個以上のジョブが連続している場合には，先行ジョブの制御ステートメントが終ったことをも意味する．

JOB ステートメントのオペランドでは，**ユーザーの会計処理ルーチン**（user accounting routine）で使用する会計情報を与えたり，ジョブの早期停止条件を与えたり，ジョブ・スケジューラーが出すメッセージを規定したりする．優先式スケジューラーの場合だと，さらに，ジョブの優先順位を与えたり，ジョブ・クラスを割り当てたり，ジョブ・スケジューラーが出すメッセージの出力クラスが要求できる．また，そのジョブで使用される主記憶装置の大きさなどを指定することもできる．

②　実行ステートメント（EXEC ステートメント）

このステートメントは，ジョブ・ステップの始まりを示し，実行されるプログラム名ないしは**カタログ式プロシージャー**（cataloged procedure：後述）の名前を指示する．また，そのステップの会計情報を与えたり，そのジョブ・ステップの実行条件を設定したり，各種情報を処理プログラムに受け渡すためのパラメータを指示したりする．優先式スケジューラーの場合には，そのジョブ

・ステップの実行制限時間を指定したり，必要となる主記憶装置のサイズを指定することもできる．

③ データ定義ステートメント（DD ステートメント）

このステートメントは，ジョブ・ステップで用いるデータ・セットについて記述し，入出力資源の割り振りを要求する．DD ステートメントのオペランドでは，データ・セット名を表示したり，ボリュームあるいに装置に関する情報を指定したり，データ・セットの属性やラベルについて記述する．

④ 指令ステートメント

このステートメントは，オペレータがシステム操作卓を用いて指令を入力する代りに，入力ジョブの流れの中に指令を入れておくために用いる．指令は，システム入出力装置の作動を始めたり，止めたり，印刷出力や操作卓への表示を求めたり，その他いくつかのオペレータ機能が達成できるようにする（詳しくは表 7-1 を参照のこと）．

⑤ 区切りステートメント（/ ＊ステートメント）

このステートメントは，入力ジョブの流れの中にデータが挿入されているときに用いる．入力データの最後に入れることによって，入力データが終りになったことを示す．つまり，データと後続の制御ステートメントを切り離す役割りを果す．

⑥ 終了ステートメント（// ステートメント）

このステートメントは，入力ジョブの終りであることを示す．したがって，ジョブ・スケジューラーがこのステートメントを判読すると，次の JOB ステートメントまで自動的に進んでしまう．

ところで，ジョブの実行のつどジョブ制御ステートメントを使ってジョブの定義するのは，かなりわずらわしい仕事である．そこで，ほとんど変更なしに繰返し使用されるような定型的なジョブ・ステップの定義は，あらかじめ直接アクセス記憶装置上の**プロシージャー・ライブラリー**（procedure library）

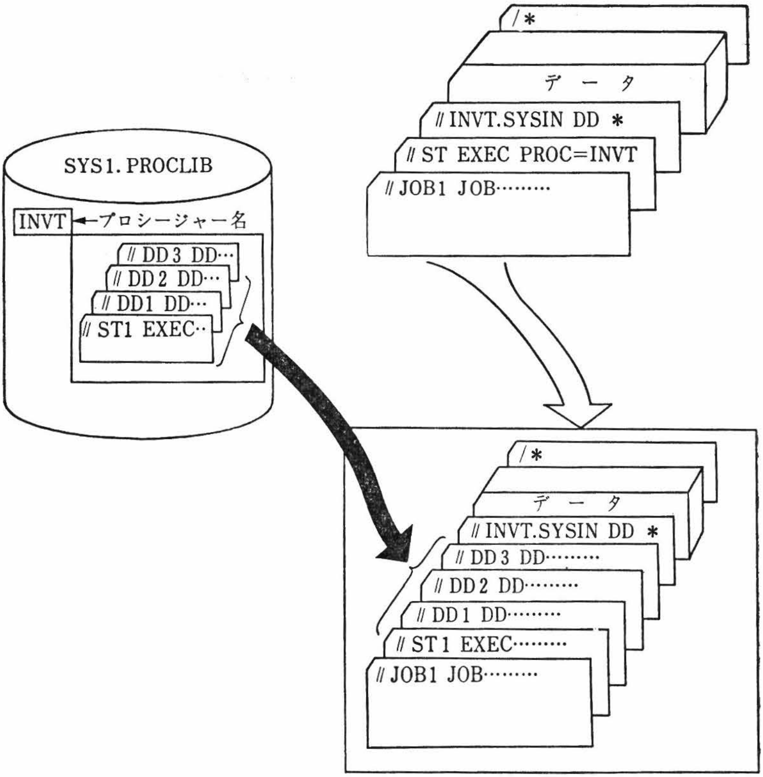


図 7-9　カタログ式プロシージャーの構想

に記憶し，カタログしておくことができる．このライブラリーは，SYS1.PROCLIB というデータ・セット名がつけられたシステム・データ・セットであり，制御プログラムがその管理に当る．また，このライブラリーに登録された一連のジョブ・ステップの定義のことを，**カタログ式プロシージャー** (cataloged procedure) と呼び，それぞれのプロシージャーには個有の名前がつけられている．この名前を**プロシージャー名** (procedure name) という (**図 7-9** 参照)．

いったんプロシージャー・ライブラリーに登録すると，その後は必要のつどEXECステートメントのオペランドでプロシージャー名を指定することで，プロシージャー・ライブラリーからそのプロシージャーを選び出してくるようにジョブ・スケジューラーに指図することができるのである．図7-9では，EXECステートメントでプロシージャー名INVTを指定することによって，この名前のついたプロシージャーを呼び込んでいる．この例では，プロシージャーは1つのジョブ・ステップで構成されているが，実際には何ステップであってもかまわない．必要なら，プロシージャー・ライブラリーから選び出したジョブ・ステップ定義の中の特定ステートメントを無効にしたり，特定ステートメントのオペランドを変更したりすることもできる．つまり，実行段階の条件に応じてプロシージャーの内容の一部を変更する自由度が与えられているのである．

このプロシージャー・ライブラリーの機能を利用して，コンピューター室で共通的に繰返し行われるタイプのジョブをあらかじめ定義しておく．こうしておけば，ジョブの実行のたびに沢山のジョブ制御ステートメントを作成する負担から解放される．それに，システムを効率的にかつ標準化された手法で使用することができるようになる．

3. マスター・スケジューラーの機能構造

マスター・スケジューラーは，オペレータの指令とメッセージとを取り扱う．指令は，オペレータが操作卓からシステムに入力するか，入力ジョブの流れの中にジョブ制御ステートメントの一部として挿入しておく（図7-10参照）．入力ジョブの流れにある指令は，読取／解釈プログラムがそれを見分けて，マスター・スケジューラーに受け渡す．マスター・スケジューラーは指令を分析したうえで，それに必要な処理を実行する．

一方，メッセージはユーザーのプログラムあるいは制御プログラムから出される．いずれの場合にもマスター・スケジューラーが中継し，システム操作卓上にメッセージの表示を行う．プログラムからオペレーターに出されるメッセージは，“オペレータへの書出し（write-to-operator)”つまり，**WTOマクロ命令**，または“応答要求つきオペレータへの書出し（write-to-operator-with-

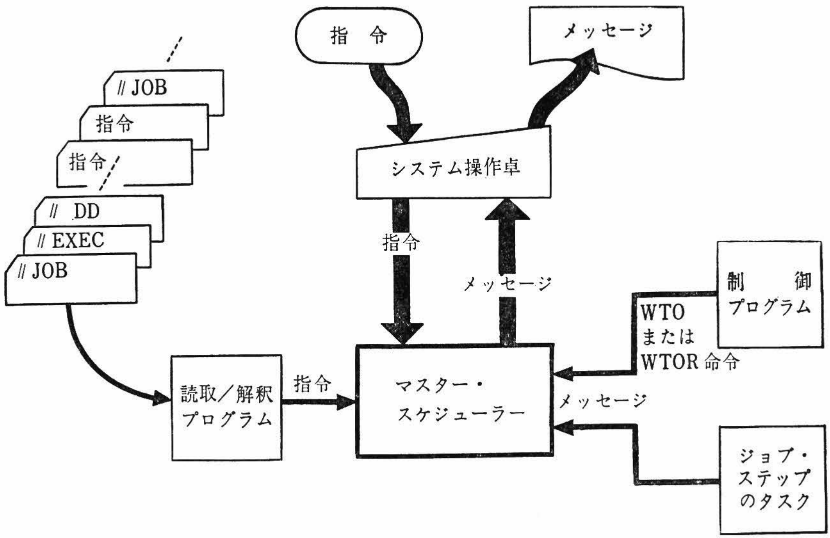


図 7-10　マスター・スケジューラーの機能構造

repley)” つまり**WTORマクロ命令**のいずれかで要請される．以上のマスター・スケジューラーの機能を図式化したのが，図 7-10 である．

オペレータはさまざまな**指令** (command) を出すことができる．これらの指令の主なものを一覧表にしたのが，**表 7-1** である．これらの指令は，つぎの 4 つのカテゴリーにまとめることができる．

① **ジョブの状況を変更する指令**：たとえば，CANCEL 指令でジョブの実行を取消したり，RESET 指令でジョブの優先順位を変更したりする．また，HOLD 指令でジョブの実行を保留にしたり，逆に RELEASE 指令で保留を解除したりするのがジョブの状況変更の指令に相当する．

② **ジョブ・スケジューラーがとる措置を変更する指令**：この種の指令としては，MODIFY 指令で開始／終止プログラムや出力書出しプログラムの制御情報を変更したり，SET 指令で入力ジョブ待行列用の装置を割り当てた

指令コード	用　　途
CANCEL	ジョブのスケジューリングないしは実行の即時停止，および出力書出しプログラムによる出力データ・セットの書出し中止のために用いる．
DISPLAY	ジョブ，待行列，データ・セット，入出力装置あるいはその他のシステムの諸活動に関する情報を操作卓上に即時に，ないしは特定の間隔で表示させるのに用いる．
DEFINE	初期値化の後で区画の大きさの変更を行なうのに用いる（特定の制御プログラムだけ）．
HOLD	ジョブを保留用待行列 (hold queue) に入れたり，1つ以上の入力クラスの待行列を保留にするのに用いる．
LOG	システム・ログ・データ・セットに情報を貯えるのに用いる．
MODIFY	開始／終止プログラムと出力書出しプログラムに関する制御情報を変更するのに用いる．
MOUNT	ボリュームの掛替えなしに，特定ボリュームを必要とする全てのジョブにある種の入出力装置の割当てを行なわせるのに用いる．
RELEASE	あるジョブを保留用待行列から解放したり，保留にされた1個以上の入力クラスの待行列を解放して，それらジョブのスケジュールができるようにするのに用いる．
REPLY	制御プログラムや問題プログラムから情報を要求するために出されたメッセージに応えるために用いる．
RESET	入力待行列，保留待行列ないしはシステム出力待行列にあるジョブのクラスや優先順位を変更するのに用いる．
SET	日時，入力ジョブ待行列用の装置あるいはプロシージャー・ライブラリーの場所などを指示するのに用いる．
START	ある種のシステム処理（読取／解釈プログラムや出力書出しプログラムなどによる処理）を開始させるのに用いる．いったん開始された処理は，STOP 指令で停止させるまで続行する．
STOP	すでに開始されているシステム処理のオペレーションを停止させたり，特定の DISPLAY 指令で要求された情報の表示を中断させるのに用いる．
UNLOAD	マウントされているボリュームを取りはずすのに用いる．
VARY	入出力装置をオンラインないしはオフラインの状態にするのに用いる．
WRITELOG	システム・ログを，出力書出しプログラムで処理するためにシステム出力データ・セット上に書き出すようにするために用いる．また，システム・ログを CLOSE するのに用いる．

表 7-1　主な指令の一覧表

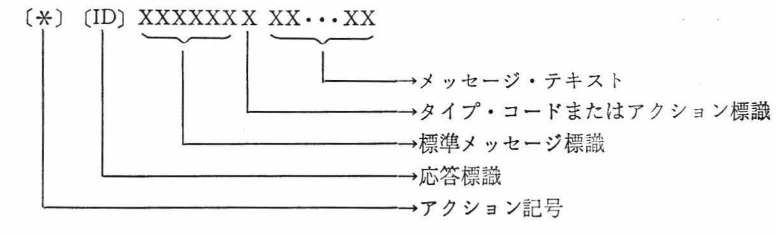
り，あるいは VARY 指令でもはや割り当てのために使用できない装置について知らせるなどがある．

③ システムあるいはジョブの状況を把握する指令：たとえば，DISPLAY 指令でジョブ，入出力待行列，入出力装置やその他のシステム諸活動に関する情報を得たり，WRITELOG 指令でシステム・ログの情報を得ることができる．

④ 情報を入力するための指令：SET 指令でそのときの日付や時刻を与えたり，LOG 指令でシステム・ログに情報を貯えたり，あるいは REPLY 指令で制御プログラムや問題プログラムからの情報要求にこたえることができる．

マスター・スケジューラーは，オペレーターに対してメッセージを出力する．メッセージには，単に情報を与えるだけに留まるメッセージとオペレータに何らかの行動を要求するメッセージとがある．メッセージは，図7-11に示すようにあらかじめ定められた形式になっている．このメッセージの形式について簡

メッセージの一般形式



メッセージの例

1. IEF403I FORTLEGO STARTED TIME=11.20.00
2. IEF281I DEVICE 193 OFFLINE
3. *IEF233A M 182, A 12345, PAYROLL
4. IEF290D WRITER NOT ACTIVE FOR CLASS Y
   REPLY 3-CHAR DEVICE NAME OR CANCEL
5. IEF234E R, 182, OLDTAP
6. IEF400W ERROR IN SCHEDULER WORK AREA
7. *00 IEF423A SPECIFY JOB QUEUE PARAMETERS
8. *01 ENTER TODAY'S DATE FOR PAYROLL CHECKS

図 7-11　メッセージの一般形式とその例

単にふれておこう．

① **メッセージ・テキスト**：この部分では，オペレータがとるべき行動あるいはシステムで生じたことなどを簡潔に表現する．たとえば，1. の例では，FORTLEGO という名前がつけられたジョブの最初のステップの実行を11時20分に開始した（STARTED）ことを示している．また，3. の例では，PAY ROLL という名前がついたジョブで，ボリューム通し番号 A12345 をもったボリュームをアドレス 182 の装置に取り付けるように要求している．

② **タイプ・コード**（または**アクション標識**）：メッセージ・テキストの直前にタイプ・コードないしはアクション標識コードが印刷される．これは，オペレータがとるべき行動を指示するためのものである．コードとしては，つぎのの 6 種類がある．単なる情報を提示するだけのメッセージの場合には，このコードは I となる．

- A――目下実行中のタスクを続行させるためには何らかの行動をとる必要があることを示す．
- D――オペレータがメッセージ・テキストの内容にもとづいて決断を下す必要のあることを示す．
- E――メッセージ・テキストで，オペレータが最終的には何らかの行動をとるように要求していることを示す．ただし，オペレータが行動をとるとらないにかかわらず，システムは処理を続行してゆく．
- I――メッセージ・テキストは単なる情報提示の目的であることを示す．
- S――重大なエラーが生じたため，処理が続行できないことを示す．
- W――これも，重大なエラーが生じたことを示す．ただし，オペレータが訂正措置をとることによって処理が続行できる．

図7-11の例をもとに，タイプ・コードからどのような行動をとるべきかを検討してみよう．1. と2. の例は，単にシステムからオペレータに情報提示が行われたにすぎない（タイプ・コードはいずれも I）．3. の場合に

は，PAYROLL ジョブを続行するためには何らかの行動，つまりボリューム通し番号 A12345 を持ったボリュームをアドレス 182 の装置に取り付ける必要がある．4. の例では，タイプ・コードがDであるので，装置のアドレスを打鍵するか，そのタスクを取消すかのどちらかの判断を下す必要がある．5. の場合のタイプ・コードはEである．したがって，メッセージ・テキストの内容から，ボリューム通し番号 OLDTAP の処理は終ったので，いずれアドレス 182 の装置上からそのボリュームを取りはずす必要があることを示している．6. の場合には，ジョブ・スケジューラーが使用している作業域でエラーが生じたので，その訂正措置がとられるまで，処理の続行はできないことを示している．7. は，3. の場合と同じである．なお，8. のメッセージはユーザー・プログラムからなされたものであり，タイプコードはない．

③ **標準メッセージ標識**：メーカーが提供するプログラムから出されるメッセージには，最低6桁の英数字コードが割り当てられている．最初の3桁でメッセージを出したプログラムを表示する．IEF はジョブ・スケジューラーからのメッセージであることを示す．IEE だとマスター・スケジューラーからのメッセージを，IEW だと連係編集プログラムからのメッセージを，IEC だとデータ管理プログラムからのメッセージを意味するといった具合いにあらかじめ定められている．

④ **応答標識**：これは，メッセージに対して応答をキーインする必要があるときだけ表示される．標識は2桁の数字からなり，応答をキイ・インする必要のある特殊なメッセージであることを示すのに使われるのである．オペレータがそのメッセージにこたえるときは，この数字を使用して応答する必要がある．

⑤ **アクション記号**：これは，オペレータが直ちにアクションを起す必要があるかどうかを示す．この個所は，＊かブランクである．＊があると，直ちにアクションをとる必要があることを示し，ブランクだとその必要がないことを示す．つまり，オペレータの注意を喚起する役目を果すのが＊記号である．

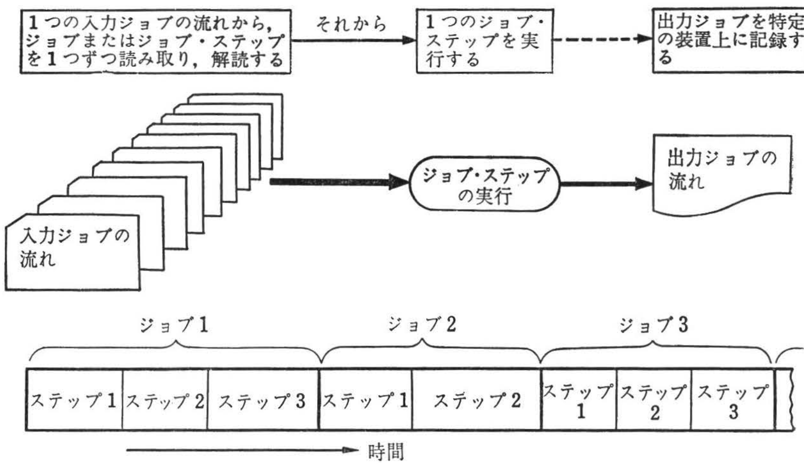


図 7-12　順次式スケジューリング・システムによるジョブ・ステップの順次処理

4.　順次式ジョブ・スケジューリング・システムの機能構造

順次式ジョブ・スケジューラーを用いたシステムを，**順次式ジョブ・スケジューリング・システム** (sequential job scheduling system) と呼ぶ．このシステムでは，ジョブおよびジョブ・ステップの連続した流れを順番に読み取って，1度に1つのジョブ・ステップを開始し，その実行を監視してゆく．したがって，ジョブは入力ジョブの流れの中で指定された物理的順序にしたがって処理される．ジョブに定められた優先順位にもとづいて処理されるということはないのである．あくまで，先着順の処理である（図 7-12 参照）．

読取／開始プログラムは，入力ジョブの流れの中から JOB ステートメントを読み取り，そのあとでジョブ・ステップの定義（EXEC と DD ステートメント）を読み取る．EXEC ステートメントのオペランドでプロシージャー名が指定されていると，入力ジョブの流れから読み取ったジョブ制御ステートメントと SYS1. PROCLIB からプロシージャー名で呼び込まれたジョブ制御ステートメントとを組み合わせる．そのあとで，制御権を開始／終止プログラムに受

<table>
<tr><td rowspan="2">常駐監視プログラム
(中核プログラム)</td><td>非常駐
監視プログラム</td><td rowspan="2">読取/開始プログラム
または
開始/終止プログラム
または
処理プログラム</td></tr>
<tr><td>マスター・
スケジューラー</td></tr>
</table>

図 7-13　順次式スケージューリング・システムの主記憶装置の利用状態

け渡す．

開始／終止プログラムは，そのジョブ・ステップを実行するために必要となる入出力装置と直接アクセス記憶装置のスペースの割り当てを行い，そのあとでジョブ・ステップの実行を開始させる．いったんジョブ・ステップが開始されると，その処理はタスク管理プログラムの指示と制御のもとで遂行されてゆく．また，補助記憶装置や入出力装置のアクセスにさいしては，いうまでもなくデータ管理プログラムの機能が関与する．ジョブ・ステップの実行が完了すると，再び開始／終止プログラムに制御権が戻される．そのジョブ・ステップの事後処理をしたうえで，次のジョブ・ステップのジョブ制御ステートメントを読み取るために読取／開始プログラムに制御権を渡す．このような繰返しで，次々と連続的にジョブ・ステップを処理してゆくのが，順次式ジョブ・スケジューリング・システムである．この順次式ジョブ・スケジューリング・システムでは，読取／解釈プログラム，開始／終止プログラムおよび処理プログラムのいずれも主記憶装置内の問題プログラム域に入ってくる（図7-13参照）．どの時点をとってみても，これら3つのプログラムのうちのどれか1つだけがロードされ実行されているのである．あるプログラムの実行が終ると，別のプログラムがロードされ，制御権が渡されて実行される．仕事が終るつど，順次的なサイクルで別のプログラムがロードされ，実行されてゆく方式をとっているのである．

これまで順次式ジョブ・スケジューリング・システムの概略を説明してきた

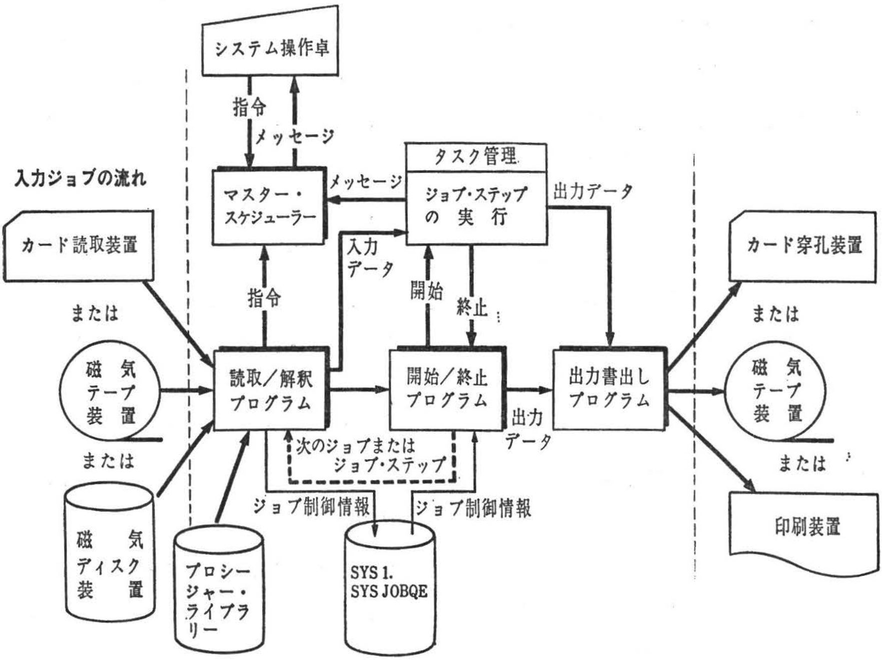


図 7-14 順次式スケジューリング・システムの構造図

が，これを機能図としてまとめると**図7-14**のようになる．この図を参照しながら，それぞれの機能について補足しておこう．

読取/解釈プログラムは，読み取ったジョブ制御カードの内容を調べて，指定上の誤りがないかどうか確かめる．これが**解釈**（interpret）**機能**である．誤りがないことが確認されると，SYS 1. SYSJOBQE データ・セット，つまりジョブ待行列用データ・セットに制御情報を集約して記憶しておく．前述したように，EXEC ステートメントでカタログされたプロシージャーを要求している場合だと入力流れのジョブ制御ステートメントとプロシージャー内のジョブ制御ステートメントとを組み合わせて，完全なジョブ定義を仕立てあげる．そのうえで，制御情報を一定の形式に集約して SYS 1. SYSJOBQE に書き出すのである．また，入力ジョブの流れの中のデータにぶっかると，そこでジョブの

流れを処理する仕事を中断し，開始／終止プログラムに制御権を渡してしまう．なお，読取／解釈プログラムで制御カードの誤りが発見されると，その表示を行ったうえで，入力ジョブの流れにある残りのカードは次の JOB ステートメントがみつかるまでバイパスされてしまう．つまり，誤りの制御カードを含んだジョブの処理を放棄してしまう．

制御権が開始／終止プログラムに渡されると，SYS 1. SYSJOBQE に一時貯えられたジョブの制御情報を取り出す．その情報にもとづいて入出力装置の割り当てや DASD 上のスペースの割り当てを行う．そのあとで，問題処理プログラムとタスクおよびデータ管理プログラムがジョブ・ステップを実行するにあたって必要とする特別な制御情報を主記憶装置の所定の場所に御膳立する (TCB や TIOT がこれに当る)．この時点で，開始／終止プログラムは，処理プログラムをロードし，その実行を監視させるために，タスク管理プログラムに制御権を渡す．処理プログラムの実行が終ると，開始／終止プログラムの終止プログラム部分に制御権が移される．終止プログラムは，データ・セットの後始末をし，使用した装置の割当てを解除する．そして，制御権は再び読取／開始プログラムにもどされる．

以上のようなサイクルが，繰返し続行される．入力ジョブの流れを読み取る装置 (system input device または SYSIN) からオペレータの介入要求の指令が発せられるか，入力ジョブの流れが終りになってしまうまで，この手続きが繰返されるのである．なお，いったん中断した処理を再開するためには，マスター・スケジューラーに対してその旨の指令を出す必要がある (START 指令で)．

開始／終止プログラムの機能は，読取／開始プログラムによって自動的に呼び出される．ところが，出力書出しプログラムの機能は，読取／解釈プログラムと一緒に開始させる必要がある．これは，やはりマスター・スケジューラーに対してその旨の指令を出すことによって達成される．入力ジョブの流れにあるデータを読み取るにしても，逆に出力データを書き出す場合にしても，順次式ジョブ・スケジューラーの読取プログラムないしは書出しプログラムは，ジョブ・ステップの実行中に要求が出されるつどレコード単位の処理を行う．言い換えれば，順次式スケジューリング・システムでは入出力データの**スプーリ**

ング (spooling) は全く行ってくれない. したがって, 読取/解釈プログラムや出力書出しプログラムに割り振られたカード読取装置ないしは印刷装置などは, SYSIN あるいは SYSOUT データを除いて問題処理プログラムで自由に使うことはできない.

以上が**順次式スケジューリング・システム**の基本的な考え方と主要な機能である.

5. 多重ジョブ処理と優先式ジョブ・スケジューリング・システムの機能構造

優先式ジョブ・スケジューリング・システムでは, 1つ以上の**入力ジョブの流れ** (input job stream) を読み取り, **優先順位** (priority) にしたがってジョブの実行をスケジュールする. このシステムでは, 相互に独立した複数個のジョブを同時に実行することができる (一般には最高15個まで). ただし, 1つのジョブを構成するジョブ・ステップは前後の依存関係があるので, 当然のことながら順番に実行されるようになっている. 優先式ジョブ・スケジューリング・システムでは, 1つのジョブ・ステップの中で必要に応じていくつでもサブタスクを発生させて (ATTACH マクロ命令で) 同時処理することもできるのが普通である. これらのタスクは, 単に同一のジョブ・ステップ内の別のタスクだけでなく, 別のジョブで生成されたタスク, 制御プログラムやオペレータが開始させたジョブ・スケジューラー用タスクなどとも同時に実行できる. このシステムで同時に処理することのできるタスクの数は, それらタスクを実行するのに必要となる諸資源の利用可能性に大きく依存している. つまり, コンピューティング・システムの規模が同時に実行できるタスクの数を左右することになる. なお, 優先式スケジューリング・システムのジョブ処理の特徴をまとめたのが, **図7-15**である.

優先式ジョブ・スケジューリング・システムが適用される場合の主記憶装置の状態は, 概略**図7-16**のような区分になる. 監視プログラムが収容される区域は, **固定域** (fixed area) と呼ばれ, 常駐部分と非常駐部分がある. 制御プログラムを構成する特別なルーチンおよび実行中の複数個のジョブで同時に共同利用されるルーチン用に**連係パック域** (link pack area) が確保される. この区

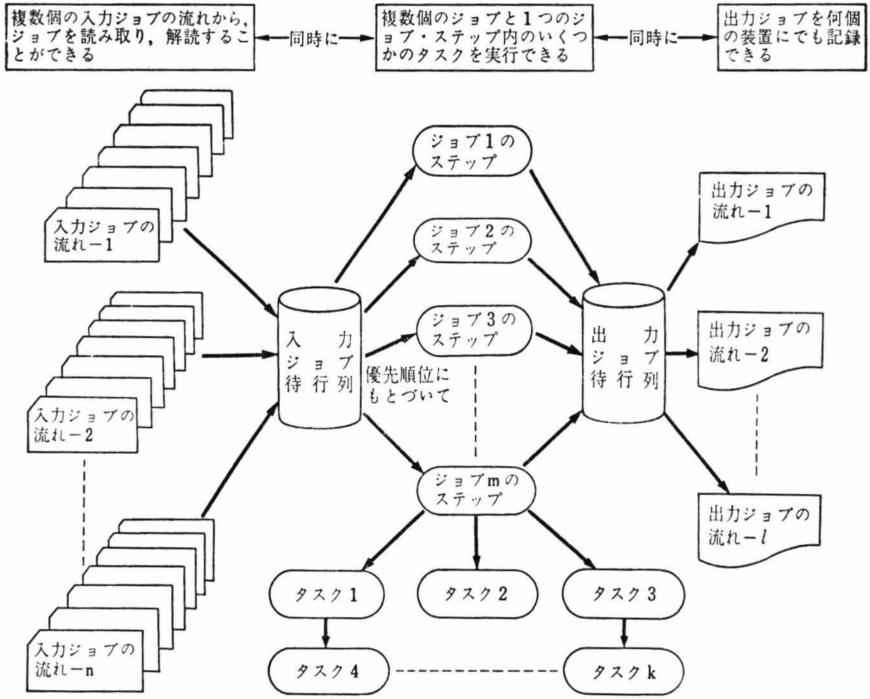


図 7-15　優先式スケジューリング・システムによるジョブ・ステップの同時処理

域に収容されるルーチンには，アクセス方式ルーチンをはじめとした制御プログラムだけでなく，ユーザーが作成した**再入可能ルーチン**（reentrant routine）も含まれる．マスター・スケジューラーは，マスター・スケジューラー区域と呼ばれる特別な場所に常駐する．

残りの記憶域は，**動的区域**（dynamic area）と呼ばれ，記憶域のプールとして利用される．制御プログラムは，実行しようとしている各ジョブ・ステップに必要な記憶域をこの動的区域の中から割り当てる．ジョブ・ステップに割り当てられた区域を特に**領域**（region）と呼ぶ．ジョブ・ステップが開始されると，処理プログラムはこの領域にロードされ，実行される．ジョブ・ステップ

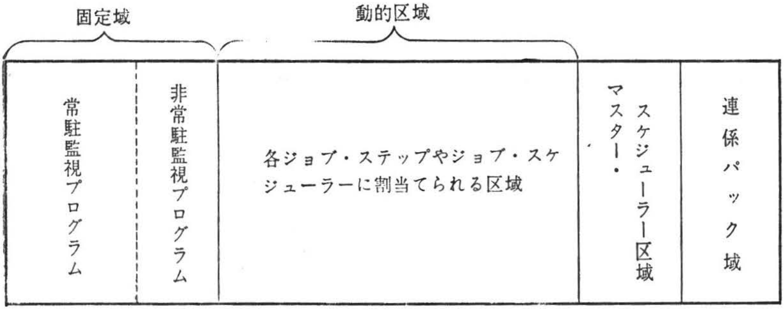


図 7-16　優先式スケジューリング・システムの主記憶装置の利用状態

が完了すると，この領域は別のジョブ・ステップの割当てに利用できるように動的区域のプールとして解放される．このように，動的区域は要求に応じて自由に分割され，ジョブ・ステップやジョブ・スケジューラーのタスクで共同利用されるのである．

1）多重ジョブ処理とスプールの構想

ジョブを構成するステップは，通常論理的な相互関係がもたらされている．そして，あるステップでの出力が次のステップでは入力として使われるような関係がある．したがって，ジョブを構成する各ステップはあらかじめ定められた順序で実行しなければならない．たとえば，代表的なジョブとして次の３つのステップから成り立っているものがある：

- 原始プログラムを目的プログラムに翻訳する．
- 目的プログラムを連係編集して，ロード・モジュールを作る．
- ロード・モジュールをロードし，実行する．

どのステップもジョブ全体に不可欠の部分である．しかも，ここに示した順序で実行しないことには意味をなさない．別の順序で行うことは考えられないのである．このように１つのジョブ内のステップは互に関係し合っている場合が多いので，あらかじめ指示された順番に実行される．ところが，ジョブは独立した仕事であるので，別々のジョブの各ステップの間には相互依存関係はな

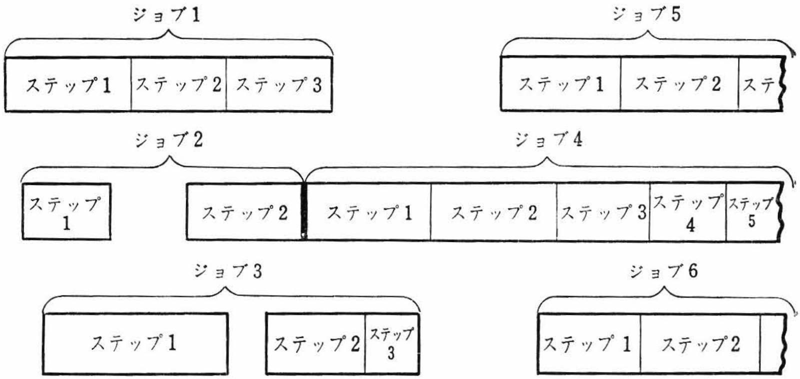


図 7-17　優先式スケジューリング・システムによる同時ジョブ処理

いのが普通である，したがって，ジョブ単位で同時併行処理を行うことは一向さしつかえないことになる．そこで，優先式ジョブ・スケジューリング・システムでは，同時に複数個のジョブを処理することができるようにしている．これが，**多重ジョブ処理**（multiple-job processing）である．これまで多重プログラミングと呼んできたのは，実はこの多重ジョブ処理のことである．

多重ジョブ処理は，システム資源が十分あることを前提としている．それぞれのジョブが同じ資源を要求することがしばしばある．同時に共有できる資源の場合にはあまり問題は生じない．しかし，同時に共有できない資源の場合だと，一方のジョブでの使用が解除されないと同じ資源を活用する別のジョブの実行は望めない．したがって，十分資源が確保されていない場合には，それぞれのジョブは，他のジョブの実行のために使用されている資源があくまで待たされてしまう．ただし，かなりのシステム資源は共同利用ができる性格をもっているので，時々待ちが生じたとしても，**図 7-17**に示しているようにジョブ全部を実行するために要する時間は，単一ジョブ処理よりは少なくてすむ．その結果，全体のスループットがかなり向上することが望める．ともあれ，多重ジョブ処理の利点を十分にひき出すためには，それなりのシステム資源が用意されていることが要求される．

さきに，同時に共同利用できる資源はあまり問題がないと述べた．しかし，全く問題がないわけではない．たとえば，DASD の場合がそうだ．DASD に格納されている特定のデータ・セットを異ったいくつかのジョブで共同利用することは許される．ただし，無条件に同時共同利用することはできない．一方のジョブでそのデータ・セットを更新している最中に，別のジョブではレコードの取り出しを行っていたら結果はどうなるであろう．正しい結果の保障はできない．そこで，無条件に共同利用ができるのは，どのジョブでもデータ・セットの内容を一切変えないような利用の場合だけに当てはまる．その他の場合はどうであろう．データ・セットの内容の変更を伴ったジョブに出合った場合には，そのデータ・セットを使用している先行ジョブが完全に終了するまで，つぎのジョブの開始を見合わせる必要がある．同様に，データ・セットの内容を更新するジョブが実行されている間は，そのジョブが完了するまで，同じデータ・セットを用いる後続のジョブを一切開始させない保障が必要だ．この2つのことが守られるならば，データ・セットを共同利用する上で生じる問題が解消できる．これをユーザ・プログラムでコントロールすることはわずらわしい．当然多重ジョブ処理においては，制御プログラムでこの面倒をみる必要がある．

多重ジョブ処理の妙味は，ジョブが優先順位にもとづいて処理できることにある．緊急を要する仕事ははやく処理し，そうでない仕事は後回しにする．これは，われわれが日常生活で普通に行っていることである．これと同じ構想をシステムに導入したいと誰しも思うであろう．この構想が反映されているのが，優先式ジョブ・スケジューリング・システムであることは言うまでもない．あらかじめジョブに与えられた優先順位にもとづいて，優先順位の高いジョブに便宜を計って，優先順位の低いジョブの開始や実行を遅らせることができる．詳細は後述することにして，その概要を説明しよう．図7-18で示したように入力ジョブ待行列には複数個のジョブ定義が読み込まれている．これは，読取／解釈プログラムの仕事である．この中から優先順位の高いジョブの順に選択して，ジョブの開始を行う．この面倒は，開始／終止プログラムがみる．ジョブが開始されて実行段階に入ると，ジョブの選択に用いられた優先順位が

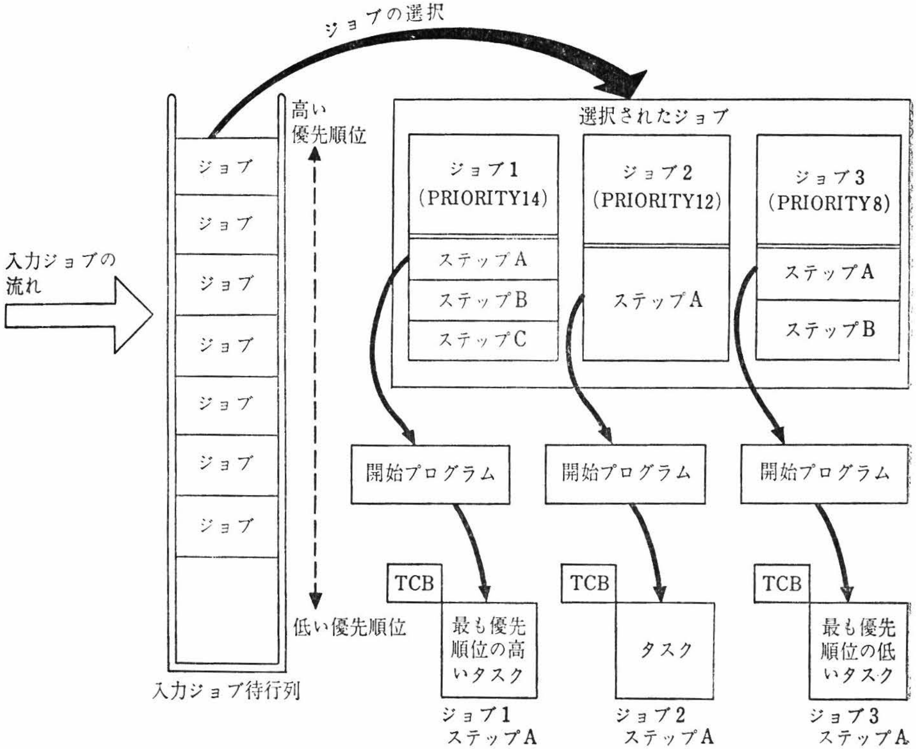


図 7-18 優先順位にもとづいた多重ジョブの開始

そのまま**タスクの実行の優先順位**(task dispatching priority)に適用される．つまり，CPU を利用する優先順位がこれで定められるのである．したがって，優先順位の高いタスクは迅速に処理されるが，そうでないタスクは優先順位の高いタスクのあい間をぬって実行されてゆく．その結果，優先順位の高いジョブははや目に選択され，短時間に処理が完了するように仕組まれている．

多重ジョブ処理は，仕事の量が非常に多く，かつシステム資源の十分なコンピュータ室に向いている．この方法を採用すると，大きなジョブでも小さなジョブでも効率良く実行することができる．また，かなりのシステム資源が共同利用できることから，それだけ資源の有効利用が計れ，全体としてのスループ

ットの向上に役立つ．しかも，優先順位にもとづいた処理ができるから，ジョブの緊急度に応じたターンアランド時間が期待できる．

さて，優先式ジョブ・スケジューリングにおける**スプール**（SPOOL）の構想について検討しておこう．

入力ジョブの流れを構成するジョブ定義および入力データは，通常穿孔カードの形式をとる．一方，ジョブの出力データは最終的には印刷形式または穿孔カード形式にする．この場合，カード入出力や印刷操作は比較的速度が遅く，時間を要する．そこで，多くの場合，図7-19に示すような方法で，入力ジョブの流れを穿孔カードから磁気テープに変換し，最終出力結果であるジョブ出力は磁気テープに記録したうえで，そのテープからカード出力ないしは印刷出力を得る．これらのデータ変換を行うためにオフラインの専用コンピュータを使用する．この方法は，中央の大型コンピュータを低速入出力オペレーションの拘束から解放することによってジョブの処理効率を高めることを主要なねらいにしたものである．このような方法を採用したコンピュータ室では，一般につぎのような手順でジョブ処理を行なう：

① 通常ジョブの優先順位を想定し，それにもとづいて入力ジョブの流れを構成する．

② 一定数量のジョブからなる入力ジョブの流れが出来上った時点で，それを磁気テープに変換する．

③ ジョブ入力テープにもとづいて，中央のコンピュータでジョブを連続的に処理する．

④ 中央のコンピュータでは，印刷出力や穿孔出力は一切行わない．その代り，印刷出力データや穿孔出力データはすべて磁気テープ上に記録する．

⑤ 印刷出力データと穿孔出力データが記録されているジョブ出力テープをオフラインで印刷形式ないしは穿孔カード形式に変換する．

⑥ 印刷および穿孔カード出力を手作業で使用目的に応じたクラスに分類し，適切な部所に配布する．

この処理手順には，随所に余分の人手が介入する．それだけ，処理全体に余

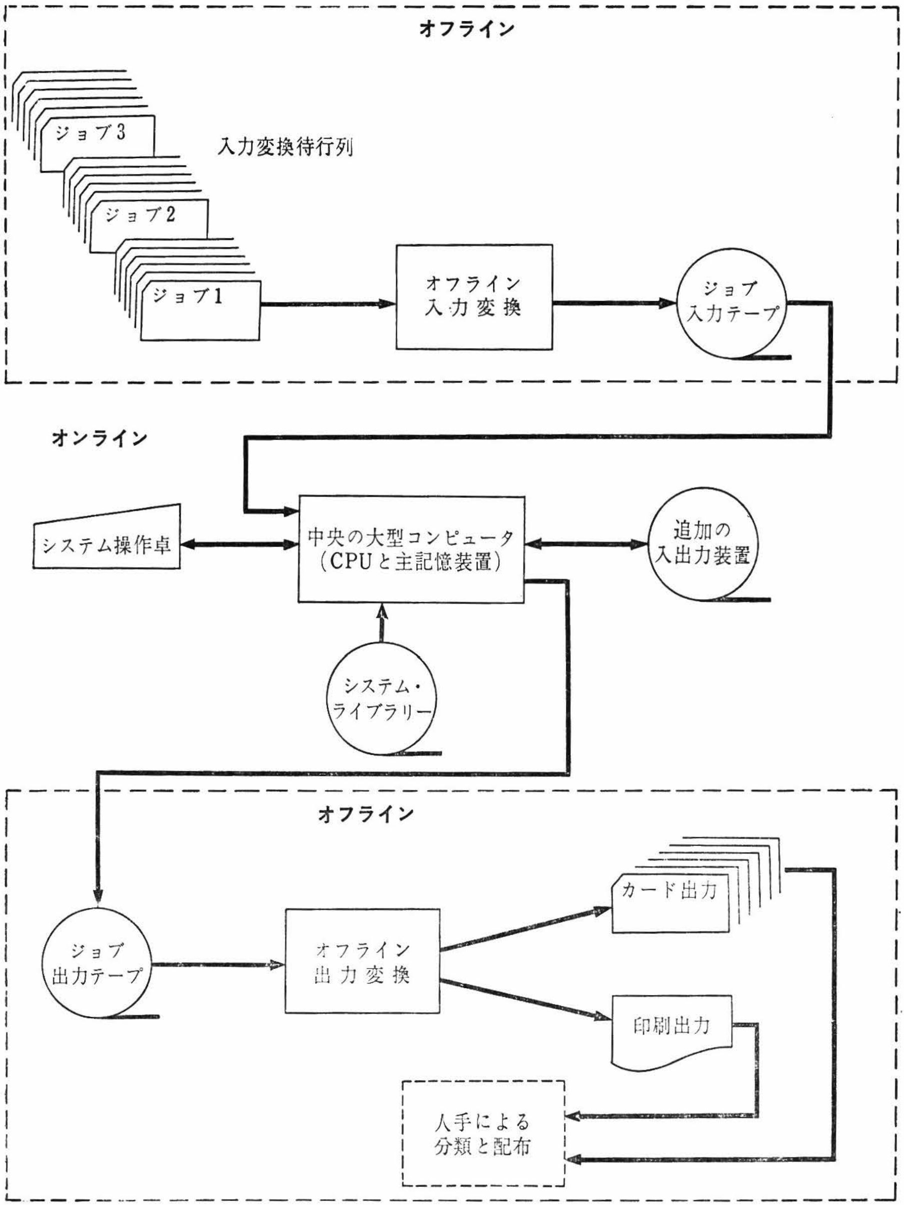


図 7-19　オフラインによるジョブ入力テープの作成とジョブの最終出力

計の時間と労力がかかる．それに，誤りの生じる可能性も高い．また，このような作業工程だと，優先順位の高いジョブを割り込ますことがむずかしくなる．ないしは，大巾に作業スケジュールを変更しないと対処できない．何よりも問題となるのは，小型のコンピュータであるにしても一台余分のコンピュータが必要になる．つまり，様々な問題を持っているのである．当然のことながら，これだけの作業を1台のコンピュータで処理し，かつ，人手の介入を可能なかぎり排除するようなアプローチが志向される．ここにも，スプールの手法を採用すべき理由が存在する．本章の最初で，多重プログラミングのもとでの同時共有不可能なシステム資源の有効利用の発展過程でスプールの出現を紹介した．このスプールが出現するもう一方の背景には，オフラインの周辺オペレーションの問題もあったのである．

ところで，優先式ジョブ・スケジューリング・システムでスプールの構想がどのように実現されているか言及しよう．このシステムでは，入力ジョブの流れの読取りとジョブ出力データの印刷ないしは穿孔作業は，それぞれ独立したタスクとして，他のジョブ処理と併行して同時に行えるようになっている．その結果，ジョブは1台のコンピュータで初めから終りまで自動的に処理することができる．これによって，オフライン・システムを用いた場合にみられる人手の介入とか時間的ずれは排除され，ターンアランド時間は大巾に短縮されるようになっている．

優先式ジョブ・スケジューリング・システムでは，一度に1つ以上の入力ジョブの流れを読み取ることができる（図7-20参照）．それぞれの入力ジョブの流れの読み取りを担当する別々の読取／解釈プログラム用タスクが機能する．入力ジョブの流れの中にあるジョブ制御ステートメントとデータはふり分けられる．前者は，一定の形式のジョブ制御用のブロックにまとめ上げられたうえで，DASD 上の入力ジョブ待行列に入れられる．一方，データもどんどん読み取られ DASD 上の入力データ・セット域に記録される．これは図7-19のオフラインの場合の入力変換に相当する作業である．

この入力ジョブの流れの読み取り作業と併行して，他の領域ではジョブ・ステップの開始と実行が同時に行われてゆく．すでに入力ジョブ待行列に入って

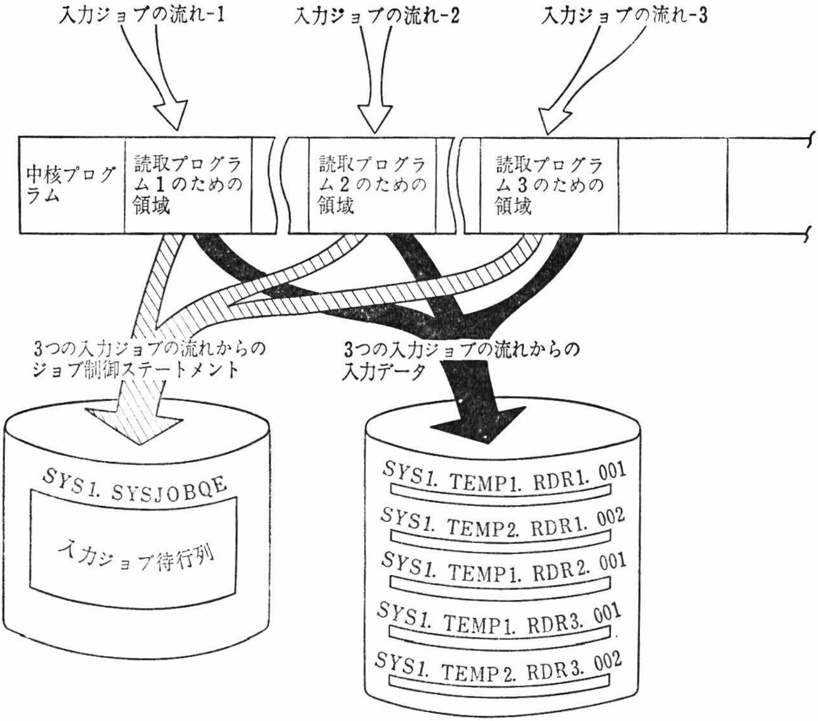


図 7-20　複数個の入力ジョブの流れからの同時読込み

いるジョブ定義のうちから，優先順位の高いジョブが選択され，開始される．この機能を果たすのは，開始／終止プログラム用タスクである．ジョブの実行が開始されると，あらかじめ DASD 上の入力データ・セットに移されている入力データが活用される．それぞれのジョブで用いる入力データが DASD 上から読み取られ処理が行われる．印刷装置や穿孔装置に出力されるデータは，直接それらの装置には書き出されず，いったん DASD 上の出力データ・セットに記録される（図 7-21 参照）．ジョブ・ステップの実行が終了すると，開始／終止プログラム用タスクは，出力データ・セット上に記録した出力データを書き出すための出力ジョブのための制御情報を作りあげ，出力ジョブ待行列に記録しておく．

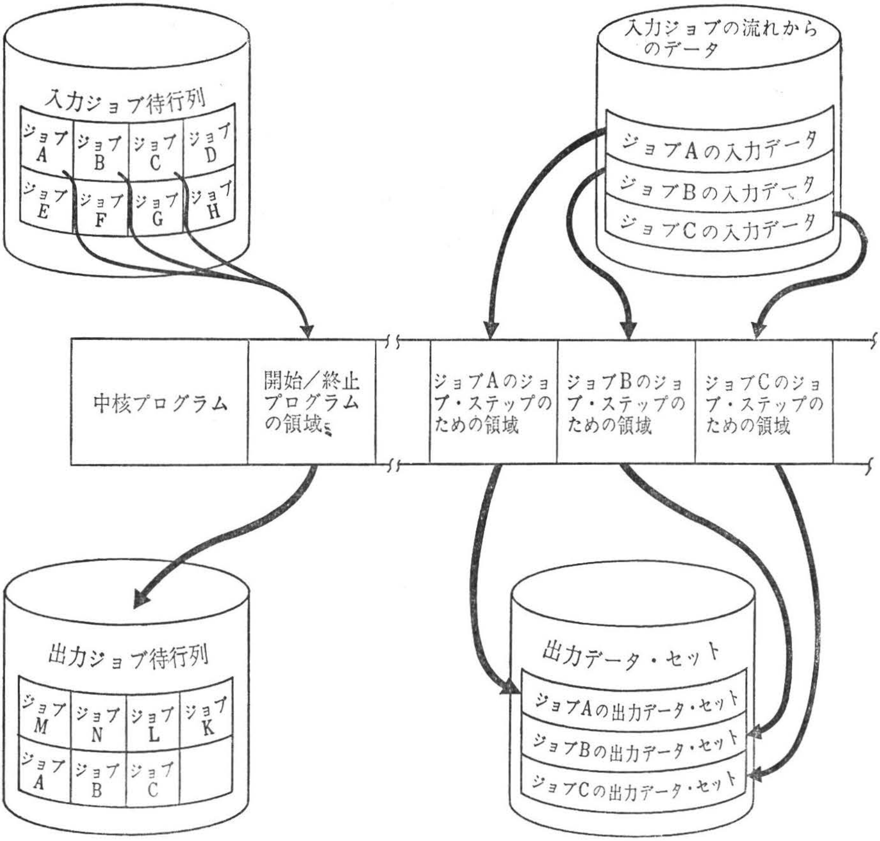


図 7-21　複数個のジョブ・ステップの開始と実行

さらに，入力ジョブの流れの読み取りとジョブ・ステップの処理に併行して，出力データ・セットの書き出し作業が行われる（図7-22参照）．この作業のために機能するのが出力書出しプログラム用タスクである．出力書出しプログラムも同時に複数個働かせることが可能である．ともあれ，出力書出しプログラムは，出力ジョブ待行列に入れられているジョブのうち優先度の高いものから順に選び出し，DASD上の対応する出力データ・セットを最終結果として印刷装置ないしはカード穿孔装置上に書き出してゆく．

このように，図7-20，図7-21，図7-22で示している機能が同時併行的に進ん

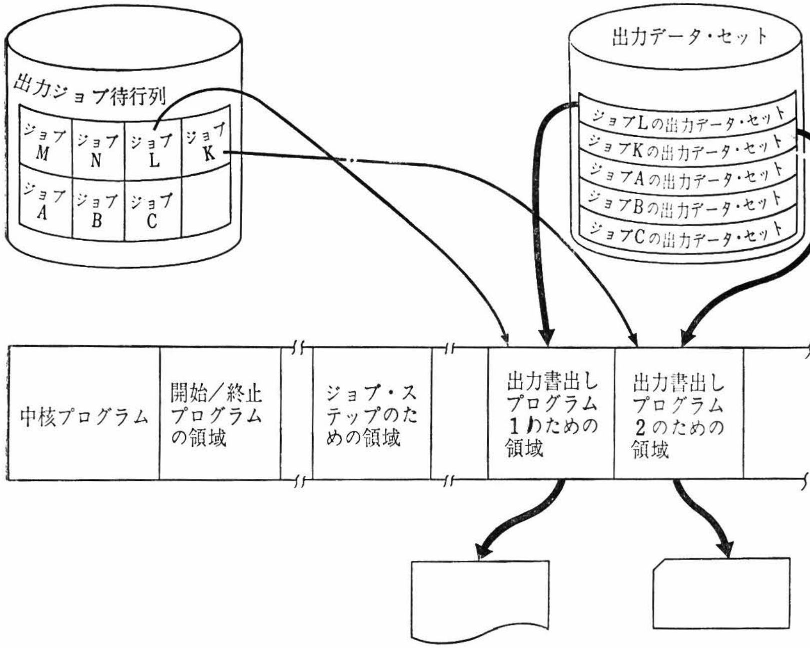


図 7-22　出力データ・セットからの書出し

でゆく．つまり，ジョブ・スケジューラーは1つまたはそれ以上のジョブ入力の流れからジョブ制御ステートメントと入力データを読み取ったり，1台またはそれ以上の出力装置にジョブ出力データを記録する．しかも，ジョブ・ステップの実行と同時併行的に行ってくれる．このような方式で，スプール機能を達成しているのである．

2)　優先式ジョブ・スケジューリング・システムの機能構造

優先式ジョブ・スケジューラーは，読取/解釈プログラムと出力書出しプログラムの機能の仕方が順次式ジョブ・スケジューラーとは異る．順次式ジョブ・スケジューラーのように，読取/解釈プログラムが開始/終止プログラムを呼び込むようなことはしない．オペレータが指令を用いて，開始/終止プログラムを開始させるのである．

優先式ジョブ・スケジューリング・システムでも，読取／解釈プログラムが入力ジョブの流れを読み取り，その流れの中のジョブ制御ステートメントとカタログ式プロシージャーの制御ステートメントを組合せ，構文上の誤りを検査して正しければ入力ジョブ待行列に制御情報を集約する．ただし，単一のジョブないしはジョブ・ステップに関する処理が終ったら，すぐに開始／終止プログラムに制御権を渡してしまうようなことはない．入力ジョブの流れがつきるまで，各々のジョブの制御ステートメントを読み取り，解読してゆく．それぞれのジョブは，入力ジョブ待行列の中に置かれてゆく（図 7-23 参照）．

さて，ここで入力ジョブ待行列について言及しておこう．入力ジョブ待行列

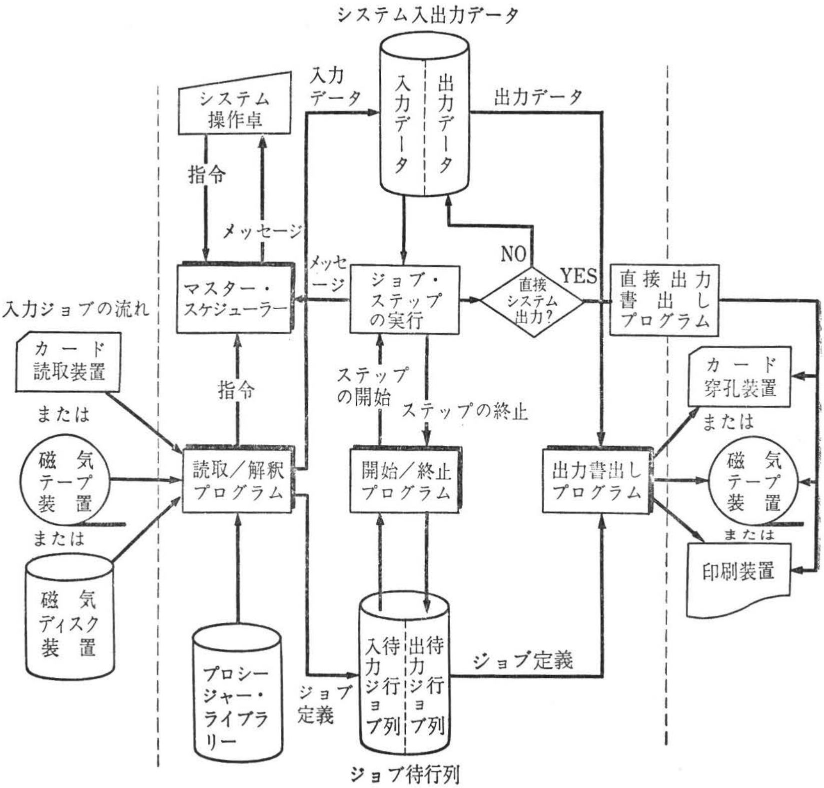


図 7-23　優先式スケジューリング・システムの構造図

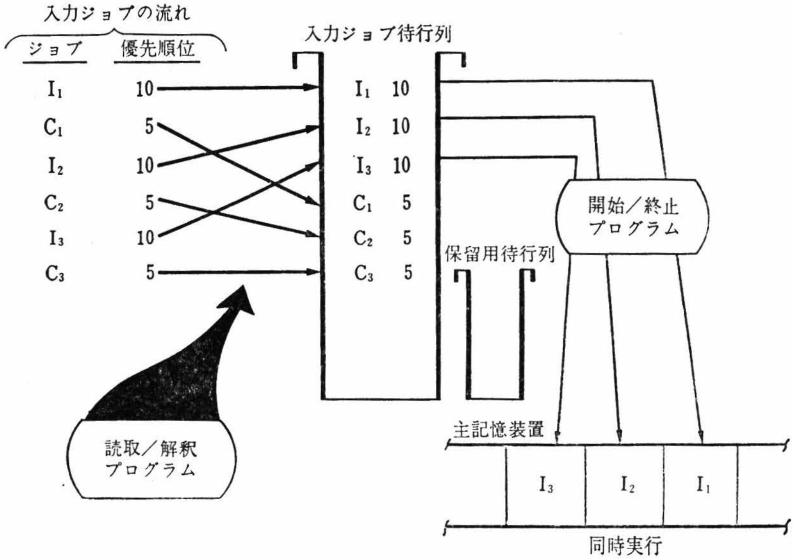


図 7-24　単一ジョブ・クラスの場合の処理

は**図 7-24** のように1個だけ用意することもあるが，一般には**図 7-25** のように複数個設けられる．各ジョブは，JOBステートメントの中のパラメータで指定したクラスと優先順位とにもとづいて振り分けられ，特定の待行列に入れられる．クラスの選択は，ジョブが必要とするシステム資源や特定のジョブ・ステップと併行して実行されると想定される他のジョブ・ステップなどを勘案して定められる．ジョブをグループ化し類別したうえで，**ジョブ・クラス**（job class）を割り振る．そして，読取／解釈プログラムに読み取らせて，該当するジョブ・クラスの待行列に入れさせるのである．一般に，15個までのジョブ・クラスと16個の入力ジョブ待行列が許される．入力ジョブ待行列のうちの15個が特定のジョブ・クラス用のものであり，英字AからOまでのクラス名が与えられる．一方，16番目の入力ジョブ待行列は**保留用待行列**（hold queue）と呼ばれる．この待行列の目的は，各ジョブ・クラスのジョブのうち一時的にその実行を見合せる必要のあるジョブをまとめて収容しておくためのものである．ジ

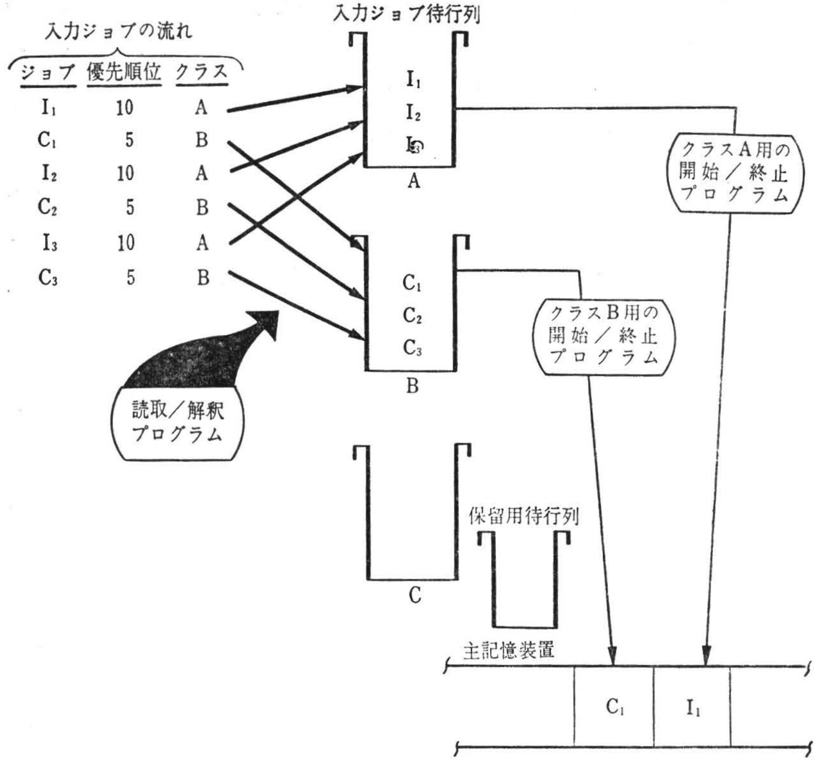


図 7-25 複数ジョブ・クラスの場合の処理

ョブは，JOB ステートメント上の情報あるいは，オペレータ指令（HOLD 指令）によって保留用待行列に入れられる．逆に保留用待行列からジョブを解放し，ジョブ・クラスと優先順位にもとづいた実行を行わせるためには，RELEASE というオペレータ指令を用いる（**表 7-1** を参照のこと）．ジョブが解放されると，必然的に特定ジョブ・クラスの待行列に入れられる．

ジョブ・クラス以外に特定ジョブ・クラス内でのジョブの実行順位を決めるために使われるのがいうまでもなく**優先順位**（priority）である．優先順位もジョブ・クラスと同様 JOB ステートメントで与えられる．一般には 15 種類の優

先順位があるが，ジョブ制御ステートメントで利用できる優先順位のレベルは0から13である．数字が大きいほど優先順位が高いことを意味する．読取／解釈プログラムが入力ジョブを処理する順番がどうであれ，JOBステートメントで指定された優先順位にもとづいて，特定ジョブ・クラスの入力待行列の中でのジョブは優先順位順に配列される．入力ジョブの流れの中に同じ優先順位とクラスを持ったジョブが複数個ある場合には，それら同一優先順位のジョブに関しては，読み取られた順番にそのクラスの待行列内に配列される．なお，入力ジョブ待行列にあるジョブの優先順位は変更することが可能だ．これは，**RESET** と呼ばれるオペレータ指令で行う．

さて，再び図7-23にもどろう．入力ジョブの流れに入力データがあると，読取／解釈プログラムはデータ管理機能を用いて DASD 上に一時的なデータ・セットのスペースを割り当て，このデータ・セット上に入力データを収容しておく．さらに，そのデータ・セットの所在を示すために特別な制御情報を修正して入力待行列の一部に入れておく．

開始／終止プログラムは1つの独立したタスクを形式しており，オペレータの指令（START 指令）がマスター・スケジューラーで処理されることによって，作動を開始する．開始／終止プログラムは，各クラスの待行列の中から最も優先順位の高いジョブを選択する．ただし，そのジョブに関して必要なすべての情報が読取／解釈プログラムによって待行列に入れられたあとでないとジョブの開始をスケジュールできないことはいうまでもない．ジョブが選択されると，そのジョブ内のジョブ・ステップを1個づつ順番に選択し実行に移らせる．ジョブ・スケジューリングの順序は，特定の開始／終止プログラムがサービスをする入力ジョブ待行列の中の制御情報にもとづいて行う．開始／終止プログラムがジョブ・ステップを開始させるときの機能は，順次式ジョブ・スケジューラーの場合と同じである．なお，同時に複数個の開始／終止プログラムがジョブのスケジューリングのために作動し，特定ジョブのジョブ・ステップを選択し，開始させることができる．図7-25に示しているように，特定ジョブ・クラスだけのサービスを行う開始／終止プログラムを設けることもできるし，複数個のジョブ・クラスに対してサービスを行うようにすることも可能だ．

優先式ジョブ・スケジューリング・システムでは，**出力流れ**(output stream)に入れたいデータ・セットのためのDASDスペースの割り当てが行なわれる．開始/終止プログラムの終止プログラム部分は，ジョブ・ステップの実行が終ると，出力ジョブ待行列に出力流れのデータ・セットに関する制御情報を貯える役割りを果す．ジョブ・ステップではなく，ジョブそのものの実行が完全に終ると，これらの出力データ・セットの処理が出力書出しプログラムで行える状態になる．これら出力データ・セットに関する制御情報は，出力データ・セットに対応するDDカードに指定された情報にもとづいて出力ジョブ待行列に配列される．出力ジョブ待行列にも，やはりクラスを設けることができる．36種類までの**出力クラス**(output class)を設けることが可能であり，各クラスを区別するために英字AからZまでと数字0から9までの36個が適用される．クラス別にする目的は，データ・セットの最終出力装置（つまり，印刷装置やカード穿孔装置あるいは磁気テープ装置など）を区別してスケジュールすることにある．なお，DASD上の出力データ・セットを優先順位にもとづいて最終出力装置に転送する働きをするのが出力書出しプログラムであることはすでに再三指摘した通りである．

ジョブ・ステップの実行中に問題処理プログラムでWRITE命令が出されるつど印刷装置やカード穿孔装置にレコードを直接出力するような出力書出しプログラムもある．この出力書出しプログラムは特に**直接システム出力書出しプログラム**(Direct System Output writer：略して**DSO出力書出しプログラム**)と呼ぶ．これは，問題処理プログラムが実行されているときに同時に機能することから，**同期的書出しプログラム**(synchronous writer)とも呼ばれている．各々のDSO出力書出しプログラムは，ジョブ・クラスの特定範囲内の処理プログラムから出力される特定クラスに対してだけのサービスしか行わない．最低1つのジョブ・クラスから最大8つのジョブ・クラスの範囲までの出力に対してサービスする．それぞれのDSO出力プログラムは，共有不可能な資源であるので，それにかかわりを持つジョブが完了するまでは他のジョブの出力を行うことはできない．あるジョブの特定クラスの出力だけの面倒しか一度にはみられないからである．

以上が優先式ジョブ・スケジューリング・システムの機能構造であり，それを図式化したのが図7-23である．なお，読取／解釈プログラムは一般には常駐プログラムとされており，何時でも機能してくれる．このプログラムは，ジョブ制御ステートメントを読み取って解読し，後続処理のためにDASD上にSYSINデータ・セットを収容する．しかし，ジョブ制御ステートメントを解釈する仕事は，読取／解釈プログラムで要する全体の時間のうちのごく一部分しか使わない．それにもかかわらず解釈プログラム部分も常駐しているのでは，主記憶装置に無駄が生じてしまう恐れがある．そこで，ジョブ制御ステートメントの解釈を行う仕事とSYSINデータ・セットからジョブ制御ステートメントを読み込む仕事とを分離することによって，主記憶装置を節約する方法が考えられる．2つの機能を分離してしまうと，読取／解釈プログラムの解釈プログラムの部分は，主記憶装置に常駐している必要はなくなり，一定量のジョブ制御ステートメントが集められたあとにだけ呼び込まれて機能すればよい．このように読取／解釈プログラムの2つの機能を分離させて働かせるような方法を，**自動的システム入力バッチ** (automatic SYSIN batching) と呼んでいる．

三．ジョブの実行サイクルとシステムの内部構造

オペレーティング・システムのもとでのプログラムの実行段階は，次の5つに集約できる：

① 中核プログラムを主記憶装置にローディングする．つまり，IPL (Initial Program Loading) の段階．

② ジョブ制御ステートメントを読み取って解読する段階．

③ ジョブ・ステップを開始させる段階．

④ ジョブ・ステップを実行する段階．

⑤ ジョブ・ステップを終了させる段階．

制御プログラムの中の中核プログラムがローディングされると，コンピュー

タの制御権はオペレーティング・システムに渡される. それ以降は中核プログラムを再びローディングすることなく，ジョブは連続的に処理されてゆく.

このようなシステムの流れを，ジョブ管理機能を中心にその内部構造と関連づけながら概観しておこう.

1. 順次式ジョブ・スケジューリング・システムのジョブの実行サイクル

順次式ジョブ・スケジューリング・システムのジョブの処理サイクルの流れを示したのが**図 7-26**である. この図を追いながら，順次式ジョブ・スケジューリング・システムの機能を説明してゆく.

中核プログラム (nucleus) をロードするためには，**システム常駐ボリューム** (system residence volume：略して SYSRES) がセットされている装置を指定し，システム操作卓上のロード・ボタンを押す. この操作で，IPL レコードが主記憶装置に読み込まれ，CPU の制御権が IPL レコードに移される. この IPL レコードが 2 番目の IPL レコードを読み込み，さらに残りの IPL プログラムを主記憶装置にロードする.

IPL プログラムは，システム常駐ボリューム上のボリューム・ラベルを探索して，VTOC を見つけだす. この VTOC の内容から，SYS1. NUCLEUS を探し出す. いうまでもなく， SYS1. NUCLEUS データ・セットには中核プログラムが収容されている. この中核プログラムを主記憶装置の固定域に呼び込むと同時に，SYS1. NUCLEUS の 1 部を占めている**中核初期化プログラム** (Nucleus Initialization Program：略して NIP) を動的区域に呼び込む. NIP は IPL プログラムから制御権を受け取り，中核プログラムの初期値化を行う. 初期値化の処理が完了すると，NIP が占めていた動的区域は読取／解釈プログラムとマスター・スケジューラーのうちのマスター指令ルーチンとで置き換えられる. これらプログラムのローディングは監視プログラムが行なう.

読取プログラムや書出しプログラムの開始を行うための START 指令と日時をはじめとした各種パラメータを指示する SET 指令とが出されると，その結果割り込みが生じ CPU の制御権はマスター指令ルーチンに渡される. このルーチンは START と SET 指令を処理したあと，読取／解釈プログラムに制御

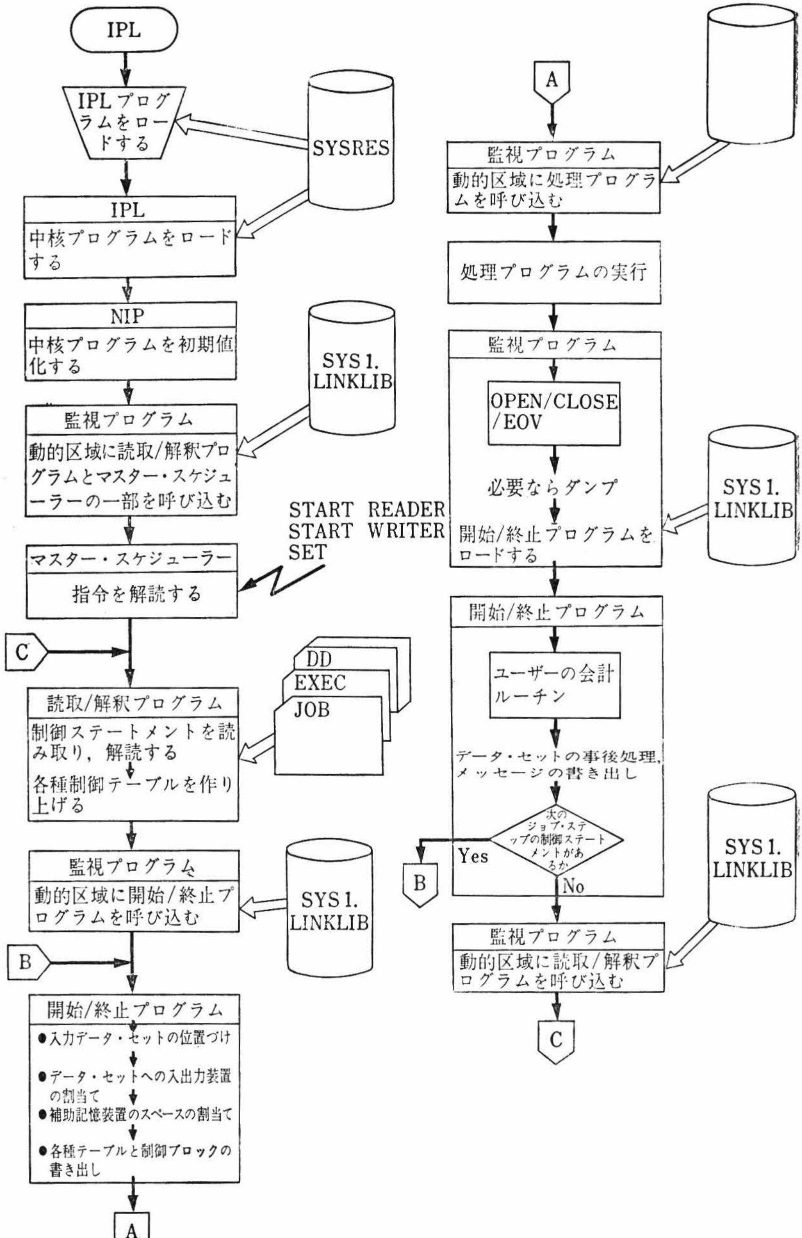


図7-26　ジョブの処理サイクルのシステムの流れ――順次式スケジューリングの場合

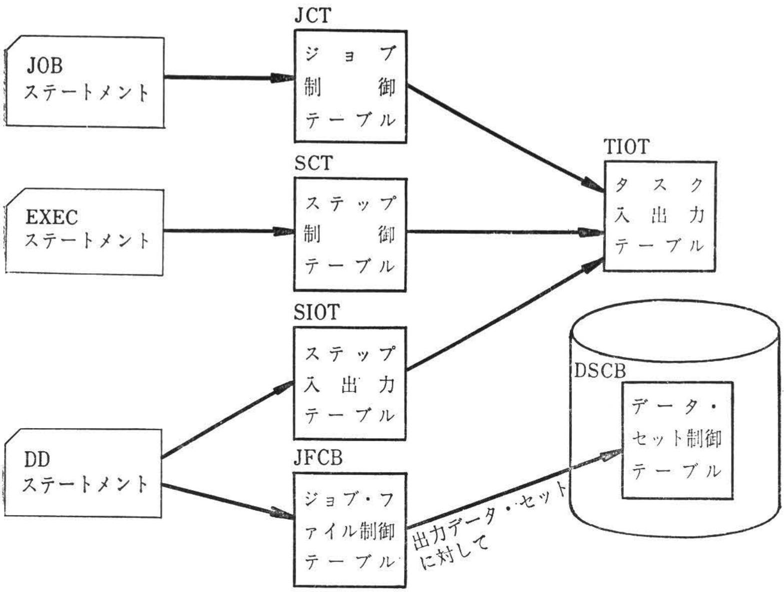


図 7-27　ジョブ・スケジューラーで生成されるブロックの相互関係

権を移す.

読取/解釈プログラムは入力ジョブの流れからジョブ制御ステートメントを読み取る. JOB, EXEC および DD ステートメントの情報がジョブ・ステップの実行に使われるのである. これらの情報は, **ジョブ制御テーブル** (Job Control Table : 略して JCT), **ステップ制御テーブル** (Step Control Table : 略して SCT), **ジョブ・ファイル制御ブロック** (Job File Control Block : 略して JFCB) および**ステップ入出力テーブル** (Step Input/Outpnt Control Table : 略して SIOT) を作りあげるのに用いられる (図 7-27参照). JCT は, JOB ステートメントの内容を一定の形式に集約したものであり, 読みとらたジョブごとに作られる. SCT は, EXEC ステートメントの情報から作られるもので, そのステップに関係した情報が集められたものである. JFCB は第 4 章ですでに説明したように各ジョブ・ステップの DD ステートメントごとに作りあげら

れ，ジョブ・ステップの実行中に対応するDD名をもったDCBがOPENされたときに主記憶装置に呼び込まれ，その情報が利用される．SIOTも各ジョブ・ステップで用いられたり，作成されたりするデータ・セットごとに作られる．

ジョブないしはジョブ・ステップの読み込みと各種制御テーブルの作成が終ると，開始／終止プログラムが動的区域にローディングされる．開始／終止プログラムは制御権を受け取って，すでに読み取りと解釈がなされているジョブ・ステップの実行開始の諸準備を行なう．読取／解釈プログラムであらかじめ準備されている各種テーブルの情報にもとづいて，つぎの諸準備を行なうのである：

① **入力データ・セットの位置づけ：** DDステートメントの情報またはカタログを通じて，それぞれの入力データ・セットが収容されているボリュームを決定する．カタログを探索する必要がある場合には，開始／終止プログラムはカタログ管理ルーチンを呼び出し，その機能を使って探索を遂行するのである．

② **入出力装置の割当て：** ジョブ・ステップで用いられる入出力装置が必要なだけ確保できないことには，そのジョブ・ステップを開始できない．したがって，開始／終止プログラムは必要な入出力装置が利用可能かどうか判断し，所要の割当てを行う．必要なら，ボリュームをのせるための指示メッセージを出す．

③ **補助記憶装置のスペースの割当て：** そのジョブ・ステップの出力データ・セットが必要とする直接アクセス・ボリューム上のスペースを確保する．この機能を果すにあたっては，データ管理プログラムの直接アクセス・ボリューム・スペース管理ルーチンの援助をあおぐ．

ジョブ・ステップの実行中に用いられるデータ・セットに関する情報を集約したJFCBは再び補助記憶装置上に書き戻される．このデータは，データ・セットがOPENされるとき，およびジョブ・ステップが終了したときの後始末のときに呼び込まれて利用される．また，**タスク入出力テーブル**（Task Input/Output Table：略してTIOT）を作りあげる．TIOTは，JCT，SCTおよび

SIOT の情報と入出力装置の割当て結果の情報などにもとづいて作られ，ステップの実行中は主記憶装置の動的区域の一部に収められている．TIOT は，OPEN/CLOSE および EOV ルーチンなどの入出力サポート・ルーチンに JFCB や割り当てられた装置などのポインターを提供する．

ジョブ・ステップの実行を開始するために必要となる全ての諸準備が終ると，開始/終止プログラムは実行すべき処理プログラムと置き換わる．つまり，監視プログラムは動的区域に処理プログラムを読み込むのである．処理プログラムは，メーカ提供のサービス・プログラムや言語翻訳プログラムであっても，ユーザー作成のプログラムであってもよい．処理プログラムは，その実行中に他のプログラムをローディングしたり，入出力オペレーションを遂行したりするために絶えず監視プログラムの援助を受けながら，処理を行う．

ともあれ，処理プログラムの実行が終了すると CPU の制御権は監視プログラムに戻される．監視プログラムは，OPEN されたままのデータ・セットがある場合には，入出力サポート・ルーチンを使ってそのデータ・セットを CLOSE する．もし，ジョブ・ステップが正常に終了したのではなく，何らかの条件で異常終止したのであるなら，主記憶装置のダンプのような特別な終了手順を踏む．

そのあとで，動的区域の処理プログラムと置き換えに開始/終止プログラムをロードし，制御権を渡す．開始/終止プログラムは，個々のジョブ・ステップを終止させ，あるいはジョブを完了させるのに必要な機能を果たす．**ユーザーの会計ルーチン** (user accounting routine) が組み込まれているなら，まずその実行を行う．さらに，入出力装置を別のジョブ・ステップで使用できるように解放し，そのジョブ・ステップの実行中に用いたデータ・セットおよび作り上げたデータ・セットの後始末を行う．この後始末を行うにあたっては，ジョブ・ステップの開始にあたって用意されたテーブルに収められているデータ・セットの事後処理に関する情報にもとづいて行う．

次のジョブ・ステップのジョブ制御ステートメントがすでに読取/解釈プログラムによって読み込まれ，解釈ずみであれば，開始/終止プログラムはそのままそのステップの開始の準備を行う．しかし，入力ジョブの流れの中にデー

ダがあると読取／解釈プログラムはそのステップのジョブ制御ステートメントだけの読み込みと解読しか行っていない．この場合には，開始／終止プログラムは，読取／解釈プログラムと置き換えられる（図7-26の(C)のポイントに戻る）．このようにして，次のジョブないしはジョブ・ステップの読取り，開始，実行および終止といったサイクルを繰り返すのである．これが順次式ジョブ・スケジューリング・システムでのジョブの実行の流れである．

なお，オペレーティング・システムのもとでジョブを実行するにさいしては，さまざまなシステム・データ・セットが関係する．これまでの説明の中にも随所でシステム・データ・セットに関する言及を行った．システム・データ・セットには，あらゆる構成のオペレーティング・システムに必須のデータ・セットとシステム構成に応じて選択可能なデータ・セットとに大別できる．ここでこれらシステム・データ・セットに関してまとめておこう．どのシステムにも必須となるシステム・データ・セットは**表7-2**に示しているように8種類ある．また，システム要件に応じて選択することのできる任意選択のデータ・セットは**表7-3**と**表7-4**にまとめてある．システム・データ・セットのうちには，**システム常駐ボリューム**（SYSRES または IPL ボリュームともいう）上に設けておく必要のあるものと，別のボリューム上に設けても差支えないものとがある．これについては，各々の表の中の説明文の中でふれているので参照されたい．

これらのシステム・データ・セットは順次式スケジューリング・システムでも優先式スケジューリング・システムでも同じように用いられる．したがって必要のつど**表7-2**，**表7-3**および**表7-4**の一覧表を参考にしていただきたい．

2.　優先式ジョブ・スケジューリング・システムのジョブの実行サイクル

1)　優先式ジョブ・スケジューリング・システムの主記憶装置の構成

制御プログラムは，それが果す機能とは無関係の特性を持っている．つまり，ジョブ管理機能，タスク管理機能あるいはデータ管理機能とは独立した特性である．この特性が制御プログラム・ルーチンと主記憶装置の使い方の双方

データ・セット名	そ の 特 徴
SYSCTLG	System Catalog. データ・セット名，そのデータ・セットを収容しているボリュームの通し番号およびそのボリュームをセットすべき装置のタイプからなるレコード群から構成される．SYSCTLG は IPL ボリュームに収容される必要があり，制御プログラムがデータ・セットをつきとめるのに用いる．
SYS 1. NUCLEUS	Nucleus (中核プログラム)．このデータ・セットには，中核プログラムと呼ばれる制御プログラムの常駐監視プログラム部分が収容されている．このデータ・セットは IPL ボリュームに設けられる必要がある．
SYS 1. SVCLIB	Supervisor Call Library. このライブラリーの構成メンバーは，制御プログラムのうちの非常駐サービス・ルーチン，データ管理のアクセス方式ルーチンおよびシステムの標準入出力装置のエラー回復ルーチンである．このデータ・セットも必ず IPL ボリュームに設けておく必要がある．
SYS 1. LINKLIB	System Program Library. このライブラリーの構成メンバーは，実行可能なロード・モジュールの形式のプログラムである．メーカー提供のプログラムはもちろんのこと，ユーザー作成のプログラムも登録できる．言語翻訳プログラム，ユーティリィティ・プログラムおよびその他のサービス・プログラムはこのデータ・セットに収容される．
SYS 1. LOGREC	System Outboard Recorder. このデータ・セットは，機械上のエラーに関する統計的データを記録するのに用いられる．記録されたデータは，エンジニアが機械の点検を行なうのに使用する．このデータ・セットは IPL ボリューム上に設ける必要がある．
SYS 1. PARMLIB	Parameter Library. このデータ・セットは，様々な構成のオペレーティング・システムが用いるパラメータや特定の値を収容する．たとえば，PRESRES や SMF (Systems Management Facility) の省略時解釈の値などがこれに相当する．
SYS 1. PROCLIB	Procedure Library. このライブラリーの構成メンバーは，プロシージャーと呼ばれる一連のジョブ制御ステートメントである．このライブラリーは，EXEC カードでプロシージャーの呼び出し要求があったさいに読取／解釈プログラムが使用する．
SYS 1. SYSJOBQE	System Job Queue. このデータ・セットは，ジョブ・スケジューラーが用いる作業域である．読取／解釈プログラムは，ジョブに関して収集した情報を一定形式のテーブルやブロックとして書き出すのに使用する．開始プログラムは，ジョブを制御するための情報をそこから得る．一方，終止プログラムは，システム出力データ・セットを出力書き出しプログラムで書き出すための制御情報をこのデータ・セットに作り出す．

表 7-2　必須のシステム・データ・セット

データ・セット名	そ の 特 徴
SYS1. MACLIB	Assembler Macro Library. このライブラリーの構成メンバーは，マクロ命令で呼び出される一連のマクロ群である.
SYS1. SORTLIB	Sort/Merge Library. このライブラリーには，ロード・モジュール形式の分類／組合せプログラム用のサブルーチンが入っている．これらのサブルーチンは連係編集プログラムで呼び出され，要求に見合った分類組合せプログラムに仕立てあげられる.
SYS1. ALGLIB	ALGOL subroutine library. このライブラリーのメンバーは，ALGOL 言語プログラム用のサブルーチンである．すべてロード・モジュールであり，連係編集プログラムが ALGOL プログラムと結合して完全なプログラムにする.
SYS1. COBLIB	COBOL subroutine library. このライブラリーのメンバーは，COBOL 言語プログラム用のサブルーチンである．すべてロード・モジュールの形式である.
SYS1. FORTLIB	FORTRAN subroutine library. このライブラリーのメンバーは，FORTRAN 言語プログラム用のサブルーチンである．すべてロード・モジュールの形式である.
SYS1. PL1LIB	PL/I subroutine library. このライブラリーのメンバーは，PL/I 言語プログラム用のサブルーチンである．すべてロード・モジュールの形式である.
SYS1. TELCMLIB	Telecommunication subroutine library. このライブラリーのメンバーは，データ通信アクセス方式で用いるサブルーチンである．すべてロード・モジュールの形式である.
SYS1. SYSVLOGX SYS1. SYSVLOGY	System log. これら2個のデータ・セットはログ・レコードを記憶するのに用いられる．ログ・レコードは，オペレーターやプログラマーが特別な指令を出したときに書き出される．このシステム・ログに記録されたログ・レコードは，一種のメッセージであり，交替勤務の終りとかあらかじめ定められた時間に印刷出力する．2種のデータ・セットを用いるのは，一方がいっぱいになったときただちにもう一方に切替えて書き出しができるようにするためである.
SYS1. ROLLOUT	ROLLOUT Data Set. このデータ・セットは，主記憶装置からロールアウトされたプログラムを記憶するのに用いる．こうすることによって，そのプログラムが占有していた記憶域をもっと優先順位の高いプログラムにあけ渡すことができる．このデータ・セットは，ロールアウト／ロールインの機能が指定されているオペレーティング・システムだけに割当て，使用する.
SYS1. ASRLIB	Automatic System Recovery library. このデータ・セットには，回復管理監視プログラムが中核プログラムの一部が壊れたと判断したときに用いられる中核プログラム・モジュールを収容している．このデータ・セットは IPL ボルュームム上に設けておく必要がある．また，特定ハードウェア・システムにだけに許される機能である.

表 7-3　任意選択のシステム・データ・セット（その1）

データ・セッ・名	その特徴
SYS 1. ACCT	User Accounting information data set. このデータ・セットは，ユーザー作成の会計ルーチンで会計情報を貯えるのに用いられる．ただし，ユーザー作成のルーチンがシステムに組み込まれていることが前提になる．
SYS 1. MANX SYS 1. MANY	System Management Facilities data set. このデータ・セットは，SMF ルーチンで収集されたデータを記録するのに使用される．2個のデータ・セットが用意されているのは，システム・ログ・データ・セットの場合と同じ理由である．
SYS 1. DUMP	System abnormal termination main storage dump data set. このデータ・セットは，制御プログラムがエラー状態を回復できない場合に主記憶域のイメージを記録するのに用いる．

表 7-4 任意選択のシステム・データ・セット（その2）

に影響を及ぼしている．

制御プログラムは，2つの作動状態，つまり**監視プログラム状態**（supervisor state）と**問題プログラム状態**（problem program state）のいずれかで作動するルーチンで構成されている．したがって，入出力命令とかある種の制御命令は，特別な制御プログラム・ルーチンの中でしか使われない．

一方，オペレーティグ・システムでの主記憶装置の構成とか使用上の割当ては，記憶保護機能とのかかわり合いが強い．記憶保護機能があることにより，主記憶装置のある区域を特定のルーチンだけに使用するように割当てることが実現できる．

制御プログラム・ルーチンが，監視プログラム状態で働くか，または問題プログラム状態で働くかは優先式ジョブ・スケジューリング・システムとの関係はない．順次式ジョブ・スケジューリング・システムの場合も同様である．説明の都合上，ここで簡単にまとめておくことにする．

制御プログラムの中のある種のルーチンだけは，監視プログラム状態で機能する．この種のルーチンでは，**特権命令**（priviledged instruction）と呼ばれる特別な命令グループの実行ができる．特権命令とは，第3章「オペレーティング・システムのためのハードウェア構造」で説明したように入出力開始，割込みの切替え，記憶保護キーの変更などのような機能を果す命令である．したがって，制御プログラムが特権的機能に対する排他的制御権を確保しているの

で，多重プログラミングの環境のもとでのプログラムやデータの完全性を保障することができているといえる．

ともあれ，監視プログラム状態で働く制御プログラムは，監視的機能とサービス的機能の両方を果す．しかも，**常駐ルーチン** (resident routine) と**非常駐ルーチン** (nonresident routine) に大別できる．

制御プログラムのうちの常駐ルーチンは，IPL 時に主記憶装置にローディングされ，プログラムの実行中は常に主記憶装置に存在する．オペレーティング・システムの他の部分でオーバレイされるようなことはない．一般には，このプログラム・ルーチンは**中核プログラム** (nucleus) と呼ばれている．中核プログラムには，ある種の非常駐 SVC ルーチンを除いた大半のタスク管理ルーチン，ジョブ管理用の特別なルーチン，入出力監視プログラムの1部および**常駐回復管理ルーチン** (resident recovery management routine) などが含まれている．中核プログラムは，いうまでもなく保護キーがゼロにセットされたプログラム状況ワード (PSW) のもとで実行される．このプログラムは，図7-28で示しているように中核プログラム用の区分データ・セット (SYS1. NUCLEUS) のメンバーとして登録されている．

中核プログラムを構成している各種ルーチンは，主として監視的機能を果すが，中にはサービス的機能を果すものもある．このサービス的機能を受け持つルーチンが，**常駐SVCルーチン** (resident SVC routine) である．SVC ルーチンは，SVC 割込みの結果利用され，制御プログラムの種々のサービスを果す．SVC ルーチンには，つぎに示す4種類があり，最初の2つのタイプが常駐 SVC ルーチンである：

- **タイプ1 SVCルーチン：** 中核プログラムの1部であり，機械チェック割込み以外のすべての割込みをマスクしてしまうようなルーチンである．
- **タイプ2 SVCルーチン：** 中核プログラムの1部であり，このルーチンの実行中に割込みが許されるようなルーチンである．
- **タイプ3 SVCルーチン：** 非常駐ルーチンであり割込み可能であるが，ルーチンは1024バイト以下の大きさである．

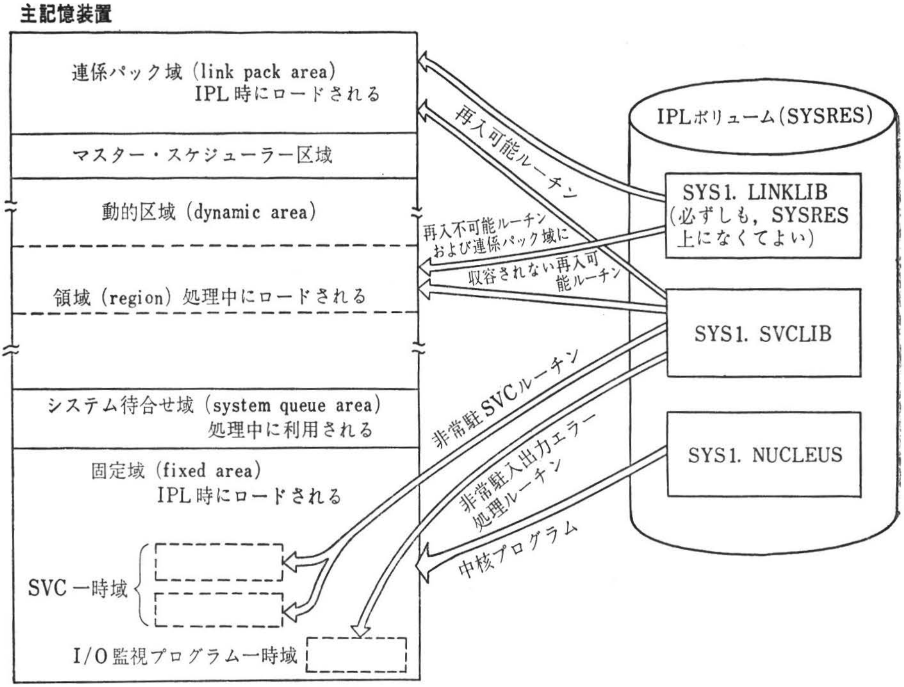


図 7-28 優先式スケジューリング・システムの主記憶装置の利用状態

- **タイプ4 SVCルーチン**： 非常駐ルーチンであり割込み可能であるが，ルーチンは1024バイト以上の大きさである．主記憶装置にローディングされる場合には1024バイト以下のセグメントに分割される．

監視プログラム状態で働く非常駐制御プログラム・ルーチンとしては，上記のタイプ3と4の SVC ルーチン，入出力エラー処理ルーチンおよび回復管理ルーチンの1部がある．

非常駐SVCルーチン (nonresident SVC routine) は，**図7-28**で示しているようにSVCLIB区分データ・セット (SYS1. SVCLIB) に収容されており，中核プログラムの中で規定されている**SVC一時域** (SVC transient area) で働

くか，**連係パック域**（link pack area）で働く．常駐ルーチンの場合と同様に記憶保護キーがゼロの PSW のもとで実行される．なお，SVC ルーチンが一時域に存在していれば，そのルーチンに対する要求があるつどそのまま利用される．新たに SYS 1. SVCLIB からローディングする必要はない．

入出力エラー処理ルーチン（I/O error handling routine）も，SYS 1. SVCLIB に収容されており，**入出力監視プログラム一時域**（I/O supervisor transient area）で働くか，連係パック域で働く．このルーチンは，入出力監視プログラムが呼び込み，入出力オペレーション中に発生したエラーを訂正したり，エラーの状態コードをアクセス式ルーチンに知らせたりする．

回復管理用の**機械チェック処理ルーチン**（machine check handler routine）も部分的に非常駐である．この非常駐ルーチンもやはり SYS 1. SVCLIB に収容されており，中核の中の一時域で実行される．機械チエック処理ルーチンは，機械チェック割込みのエラーを修復し，処理を続行するために機能する．

以上が監視プログラム状態で働く制御プログラム・ルーチンの概略であるが，問題プログラム状態で働く制御プログラムとしてはジョブ管理用のルーチンとアクセス方式ルーチンがある．これらルーチンは，連係パック域で実行されるか**動的区域**（dynamic area）で実行される．ジョブ管理用ルーチンは，しばしば中核プログラムや**システム待合せ域**（system queue area）にデータを貯えることから，当然のことながら保護キーがゼロの PSW のもとでしか機能しない．一方，アクセス方式ルーチンはそれを呼び出したプログラムと同じ PSW のもとで働く．なお，ジョブ管理用ルーチンは SYS 1. LINKLIB に，アクセス方式ルーチンは SYS 1. SVCLIB に収容されている．

さて，本題の主記憶装置の構成と使用上の割当てに移ろう．優先式ジョブ・スケジューリング・システムの主記憶装置の利用状態は，図 7-28 に示している．優先式ジョブ・スケジューリング・システムの主記憶装置は，固定域，システム待合せ域，連係パック域，マスター・スケジューラー区域および動的区域に大別されて，利用される．

固定域（fixed area）は，IPL 時に中核プログラムがローディングされる記憶域である．固定域の記憶保護キーは，言うまでもなくゼロであり，その内容

は制御プログラムでしか修正できない．図7-28で示しているように固定域の中には複数個の一時域があり，必要のつど非常駐ルーチンが呼び込まれる．一時域は先述したように中核プログラムの中で規定され，固定域の一部をなしている．一時域には，**入出力監視プログラム域** (I/O supervisor transient area) と **SVC 一時域** (SVC transient area) の 2 種類があり，それぞれ非常駐入出力エラー処理ルーチンと非常駐 SVC ルーチンとで利用される．固定域の他のルーチンと同様に一時域のルーチンも記憶保護キーがゼロの PSW のもとで働く．また，一時域で機能する全てのルーチンは，SYS 1. SVCLIB に収容されている．

SVC 一時域の大きさは，1024バイトであり非常駐 SVC ルーチンのために確保されたものである．この一時域の数は，最低 2 個必要であるが，必要に応じて数を増やすこともできる．その数は，システムの生成時に指定することになっている．ともあれ，ある種の非常駐 SVC ルーチンの処理が要求され，かつそのルーチンが一時域にも連係パック域にも存在しない場合には，該当ルーチンが利用可能な一時域に呼び込まれる．一時域のいずれも利用できない場合だと，SVC ルーチンを CALL したタスクは一時域が利用できるまで **待ち状態** (wait state) に置かれる．一時域が空になったり，そこに収容されている SVC ルーチンの実行が完了すると，その一時域がはじめて利用可能になる．

ただし，SVC一時域が利用可能状態にない場合でも，目下使用されている一時域を他の SVC ルーチンのために充当することができる．これが可能なのは，一時域を必要とするタスクが一時域を利用しているタスクよりも高い優先順位をもっている場合である．複数個の一時域がこのような状態の時には，充当される一時域は，最も優先順位の低いタスクが使用している区域ということになる．ともあれ，割当てられた一時域には優先順位の高いタスクが要求した SVC ルーチンがロードされ実行される．実行が完了すると，前のタスクのための SVC ルーチンが再びロードされ，その処理が再開される仕組みになっているのである．

入出力監視プログラム一時域は 1 個しかない．やはり1024バイトの大きさであり，入出力監視プログラムが呼び込んだ非常駐入出力エラー処理ルーチンが

収容される.

システム待合せ域（system queue area）は，固定域に隣接しており制御プログラムが作りあげるテーブルや待行列を収容するために設けられた記憶域である．システム待合せ域は，**中核初期値化プログラム**（nucleus initialization program）でお膳立てされる．この区域の記憶保護キーもゼロである．したがって，制御プログラムのルーチンでしか修正できない．ともあれ，システム待合せ域に収容された情報は，全てのタスクの状態とシステム資源の利用状態を示すのに使われる．この区域は，しばしば**システム待合せスペース**（system queue space）と呼ばれることもある．

連係パック域（link pack area）には，SYS 1. LINKLIB と SYS 1. SVCLIB に収容されている**再入可能ルーチン**（reentrable routine），SYS 1. LINKLIB に収容されている各種ルーチンの所在個所を示すトラック・アドレス（一般には，BLDL リストと呼ぶ）および SYS 1. SVCLIB に収容されている逐次再使用可能な性質がもたされている入出力エラー回復ルーチンが記憶される．連係パック域に記憶されているルーチンは，すべてのタスクから利用することが可能である．したがって，必要のつどそれぞれの区域にローディングする必要はなくなり，それだけ処理効率の向上が期待できる．また，連係パック域に BL DL リストを常駐させておくと，SYS 1. LINKLIB 上のルーチンを捜し出すのに要する時間の短縮が実現できる．なお，連係パック域のタイプ 3 および 4 の SVC ルーチンと入出力エラー回復ルーチンは監視プログラム状態で働くが，その他のルーチンはそれの呼び出しを行ったルーチンと同じ状態で機能する．つまり，問題プログラム状態のプログラムから要求された場合だと問題プログラム状態で働くし，逆に監視プログラム状態のプログラムから要求されたら監視プログラム状態で働くことになる．

連係パック域に収容すべきルーチンは，ユーザーが選択することができる．一定形式でルーチン名のリストを作成する．IPL 時に，NIP プログラムがこのリストにもとづいて主記憶装置の高位の番地から指示されたルーチンをローディングしてくれるようになっている．その様子は，図7-30に示しているので参照されたい．NIP 実行後は，再び IPL 手順がとられるまで連係パック域の内容

を変更することはできない．なお，連係パック域にはユーザー指定のルーチン以外にシステムが指示するジョブ管理用のルーチンとアクセス方式ルーチンも収容される．

動的区域（dynamic area）は，ジョブ・ステップとシステム・タスクが要求する記憶域として使われる．この区域は，連係パック域とシステム待合せ域に囲まれた主記憶域である．ジョブ・ステップやシステム・タスクが開始されると動的区域の中から必要な記憶域が**領域**（region）と呼ばれるブロック単位で割当てられる．

通常，ジョブ・ステップやタスクを構成するプログラムが要求する記憶域はあらかじめ割当てられている領域の中から割り振られる．ただし，第８章で説明しているようにロールイン／ロールアウトの機能を用いると一時的に追加の記憶域を確保することができる．領域は，前述のようにジョブ・ステップやシステム・タスクが開始されるときに割当てられ，ステップやタスクの実行が終了するまで割当てられたままである．実行が終ると，解放された領域が動的区域の利用可能なスペース部分に戻され，再び別のジョブ・ステップやシステム・タスクの主記憶域の割当てに活用されることになる．

図7-30の中で図示しているようにマスター・スケジューラー・タスク用の領域は，中核初期化プログラム（NIP）が割り当ててくれる．この領域は，連係パック域に隣接しており，中核プログラムが主記憶装置に存在する限り割り当てられたままの状態が続く．

ジョブ・ステップに割り当てられた領域の記憶保護キーは１から15のうちのいずれかである．領域内の記憶域は，ステップの実行中必要に応じて２Ｋ単位のブロックで割り振られてゆき，それぞれのブロックの保護キーも関連する値がセットされる．しかし，そのジョブ・ステップ用の領域のうちで未割り振りのブロックのキーはゼロにセットされたままである．システム・タスク用に割り当てられた領域の保護キーは，当然のことながらゼロである．また，特定領域に割り当てられていない動的区域の保護キーもゼロにセットされる．

2）優先式ジョブ・スケジューリング・システムのジョブの処理サイクル

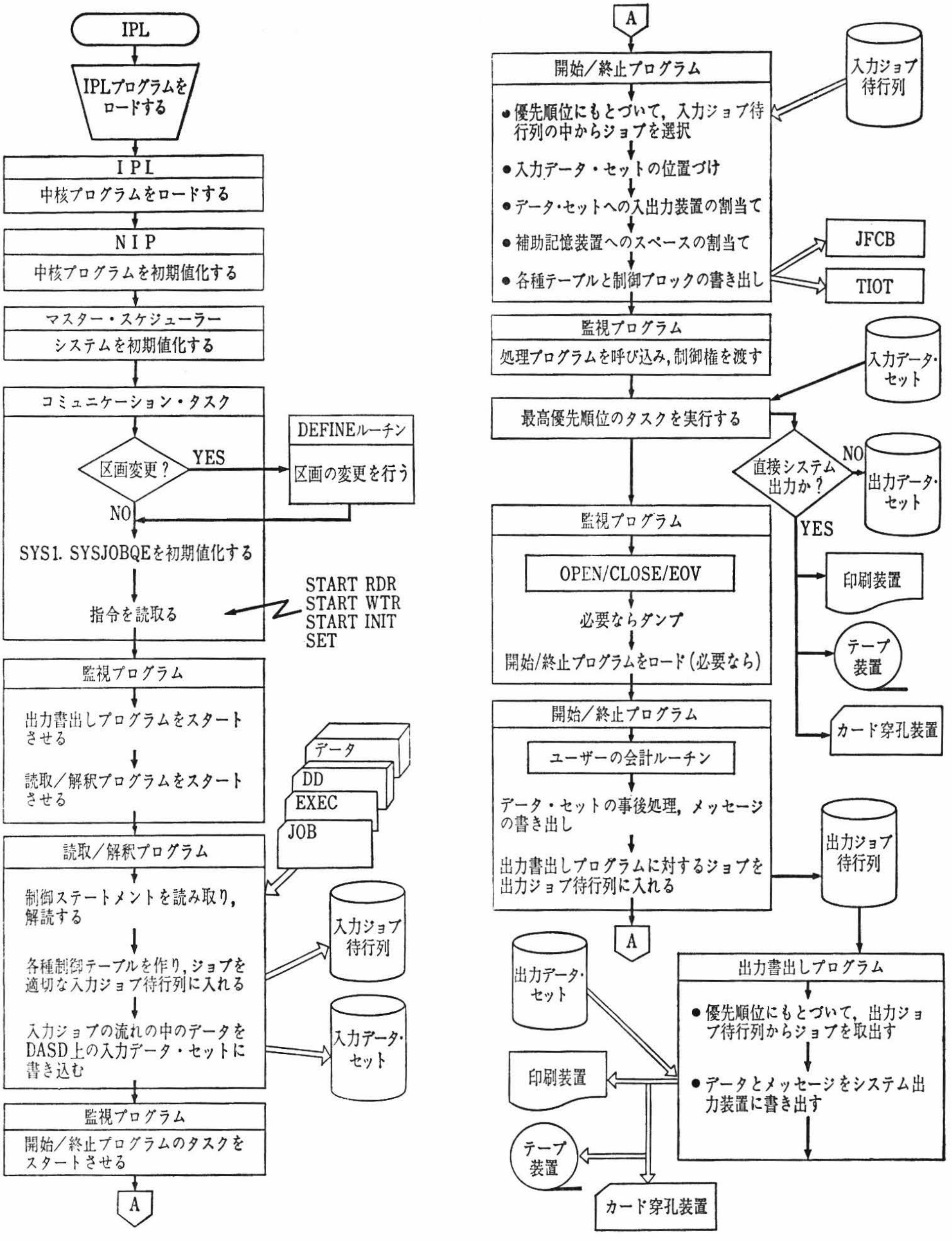


図 7-29　ジョブの処理サイクルの流れ――優先式スケジューリングの場合

優先式ジョブ・スケジューリング・システムにおけるジョブの処理サイクルを示したのが**図7-29**である．まずは，この図を追いながら，優先式ジョブ・スケジューリング・システムの機能を構造的に把握することにしよう．

オペレーティング・システムを使用するに先立って，**システム常駐ボリューム**(system residence volume)から主記憶装置に中核プログラムをローディングする必要がある．この機能は先刻説明したようにIPLプログラムとNIPプログラムとが遂行してくれる．

第5章「データ管理の諸機能とその構造」の「ボリューム・ラベルとデータ・セット・ラベル」の個所でふれたようにシステム常駐ボリュームのシリンダー0のトラック0には2個のIPLレコードが収容されている．中核プログラムをロードするためには，システム常駐ボリュームを指定したうえで操作卓上のロード・ボタンを押す．これによって，最初のIPLレコードが主記憶装置に呼び込まれ，CPU制御権が渡される．このIPLレコードは，2番目のIPLレコードを読み込む．さらに，2番目のレコードがIPLプログラムを主記憶装置に読み込む仕組みになっている．この順序は，順次式ジョブ・スケジューリング・システムの場合と変らない．IPLプログラムは，主記憶装置をクリアし，記憶保護キーをすべてゼロにセットする．そのあとで，IPLプログラム自体を主記憶域の上部に再配置し，もと占めていた記憶域をクリアする．さらに，主記憶装置の低い番地に中核プログラムをローディングする．このオペレーションが終ると制御権をNIPに渡す．IPLプログラムが実行されたあとの主記憶装置の様子を示すと，**図7-30**の①のようになる．

NIPは，中核プログラムの中にある各種テーブルを初期値化し，中核プログラムを構成する各種ルーチンとテーブルのアドレスおよび記憶域の境界を決定し，**間隔計時機構**(interval timer)のチェックと時間のセットを行う．そのうえで，システム待合せ域の境界を定め，連係パック域に指示されたルーチンをロードし，さらにはマスター・スケジューラー・タスクの領域を割り当てる．これらの仕事が終ると，マスター・スケジューラー・タスク用のルーチンに制御権を渡す．NIPが実行されたあとの主記憶装置の構成は，図7-30の②で示している．

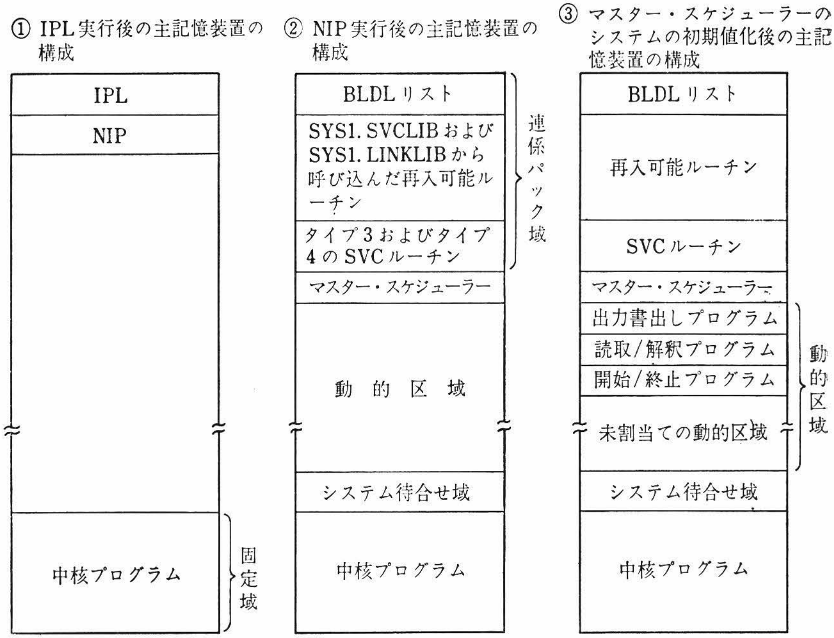


図 7-30　優先式スケジューリング・システムの主記憶装置の構成

マスター・スケジューラー・タスクのうちの初期値化ルーチンがシステムの初期値化を行ったあとで，コミュニケーション・タスクに制御権が移る．特別なシステムでは，区画の変更ができるがその要請がないかどうか調べる．必要ならその指令の処理をしたあとで，入力ジョブ待行列および出力ジョブ待行列のためのデータ・セットの初期値化を行う．また，SYS 1. LOGREC データ・セットをOPENする．そのあとで，STRAT指令およびSET指令を処理する．START 指令を処理することにより，読取／解釈プログラム用タスク，開始／終止プログラム用タスクおよび出力書出しプログラム用タスクが生成される．指令の処理が終ると制御権は中核プログラムに渡される．この時点での主記憶装置の様子は，図7-30の③に示した通りである．

監視プログラムは，読取／解釈プログラムおよび出力書出しプログラムをスタートさせたあとで制御権を読取／解釈プログラムに移す．読取／解読プログ

ラムは，順次式スケジューリング・システムの場合と同じように，ジョブ制御ステートメントを読み取って解読し， JCT, SCT, JFCB, SIOT などの各種制御テーブルを作り上げて入力ジョブ待行列に入れてくれる．優先式スケジューリング・システムの場合には，さらに入力ジョブの流れの中にある入力データを DASD 上の入力データ・セットに書き込む機能を果たしてくれる．最初の1ジョブ分の処理が終ると，一たん制御権を監視プログラムに返し，開始／終止プログラムのタスクをスタートさせる．そのあとで，後続する入力ジョブの読取りと解釈を再開する．つまり，ジョブ制御ステートメントの読取／解釈の仕事とジョブの開始，実行ないしは終止の仕事とが併行的に進行してゆくようになるのである．

実行が開始された開始／終止プログラム用タスクは，入力ジョブ待行列の中から最も優先順位の高いジョブを選び出し，その実行のための前準備を行う．ジョブの実行のための前準備の仕事は，順次式ジョブ・スケジューリング・システムの場合と変らない．入力データ・セットの場所をつきとめ，使用するデータ・セットの入出力装置の割当てと補助記憶装置のスペースの割当てを行う．そのあとで，ジョブ・ステップの実行中に用いられるデータ・セットに関する情報を集約した JFCB の内容を更新したうえで，再び DASD 上に書き戻す．また， JCT, SCT および SIOT の情報と入出力装置の割り当て結果の情報にもとづいて TIOT を作りあげて書き出す． ジョブの実行を開始するに必要な前準備が終ると，制御権は監視プログラムに戻される．

監視プログラムは，ジョブ・ステップ用タスクを実行するに必要な主記憶装置上の領域を未割り当ての動的区域の中から確保し，その場所に処理プログラムを呼び込んで制御権を渡す．処理プログラムは，必要に応じて読取／解釈プログラムが作りあげてくれた DASD 上の入力データ・セットを用いながら処理を行う．出力データに関しては，直接システム出力の指示がないものに関しては，出力データ・セットとして DASD 上に一時的に書き出す．このデータ・セットは，もちろん出力書出しプログラムで処理させるためのものである．ジョブ・ステップの実行が終了すると制御権は再び監視プログラムに戻される．

監視プログラムは，OPEN されたままのデータ・セットがある場合には，そ

のデータ・セットをCLOSEする．そのジョブ・ステップが何らかの条件で異常終止したのであれば，主記憶装置のダンプのような特別を終了手順を踏み，そのジョブを終らせてしまう．そのあとで，開始／終止プログラム用のタスクが CPU に存在しない場合だと，開始／終止プログラムをロードし，制御権を委譲する．

開始／終止プログラムは，個々のジョブ・ステップを終止させ，あるいはジョブを完了させるのに必要な機能を果す．ユーザーの会計ルーチンが組み込まれているなら，まずその実行を行う．そのステップで用いた入出力装置のうち不要なものを別のジョブ・ステップで利用できるように解放し，データ・セットの後始末を行う．また，そのジョブのために割り当てられていた領域を他のタスクで利用することができるように解放する．さらに，出力データ・セットを出力書出しプログラムで出力できるようにするために，出力ジョブに関する情報を出力ジョブ待行列に貯えておく．

ジョブないしはジョブ・ステップに関する後始末が終ると，つぎのジョブないしは後続するジョブ・ステップの実行の準備に移る．新しいジョブの場合だと，やはり優先順位にもとづいて入力ジョブ待行列の中からジョブの選択を行ってくれる．これらのジョブの開始，実行，終止といった一連のタスクは，読取／解釈プログラムのタスクとは独立して，同時併行的に遂行されてゆくのである．

出力ジョブ待行列にジョブが入ると，出力書出しプログラム用タスクを稼動させることができる状態になる．出力書出しプログラムもジョブの優先順位にもとづいて作動する．出力ジョブ待行列に複数個の出力ジョブが入っている時には，その中から一番優先順位の高いジョブを選択して，それに見合った出力データ・セットを用いて出力を行う．なお，出力書出しプログラムのタスクも他のタスクと独立的に機能してくれるので，入力ジョブの読取りと解釈の仕事，ジョブ・ステップの開始と実行の仕事などと完全併行処理が実現できるのである．

以上が優先式スケジューリング・システムのジョブの処理サイクルである．図7-29を図7-26の順次式スケジューリング・システムのジョブの処理サイクル

と対比しながら各自でまとめていただきたい.

3) 指令処理の内部構造

ジョブ管理ルーチンは,再三指摘してきたようにプログラマーとかオペレーターといった人的資源と制御プログラムとの間でのコミュニケーションの役割りを果す. この処理機能は, **指令処理** (command processing) と**ジョブ処理** (job processing) に大別できる. 前者はマスター・スケジューラーによって,後者はジョブ・スケジューラーによって達成されることは言うまでもない.

指令処理とは,オペレーターが操作卓ないしは入力ジョブの流れの中で出した指令を読み取り,スケジューリングし,実行する機能である,一方,ジョブ処理とは,ジョブ制御ステートメントを解読し,そのステートメントで定義されているジョブ・ステップの実行の準備と後始末を行い,出力データ・セットからシステム・メッセージやシステム出力データ・セットを書き出す機能のことである.

ところで,ジョブ管理ルーチンはいくつかのタスクを遂行することによって,指令処理とジョブ処理の機能を果してくれる. このジョブ管理用のタスクは,処理プログラムを実行することによって達成されるユーザー・タスクと区別するために特に**システム・タスク** (system task) と呼ばれている. システム・タスクが作られるときにも主記憶装置の領域が割り当てられる. このタスクは,連係パック域に収容することも可能であるが,そうでない場合だと確保された領域にローティングされ,その場所で作動する. システム・タスク用の領域の記憶保護キーは,いうまでもなくゼロである. したがって,システム・タスクを遂行するルーチンは,保護キーがゼロの PSW のもとで働くのでその領域だけでなくシステム待合せ域にも書き出しを行うことができる.

さて,ここでは指令処理用のシステム・タスクを中心にシステムの構造を説明してゆこう.

指令の処理には,つぎの3つの局面がある:

● 指令の読取り (コンソール・コミュニケーション・タスクまたは読取/

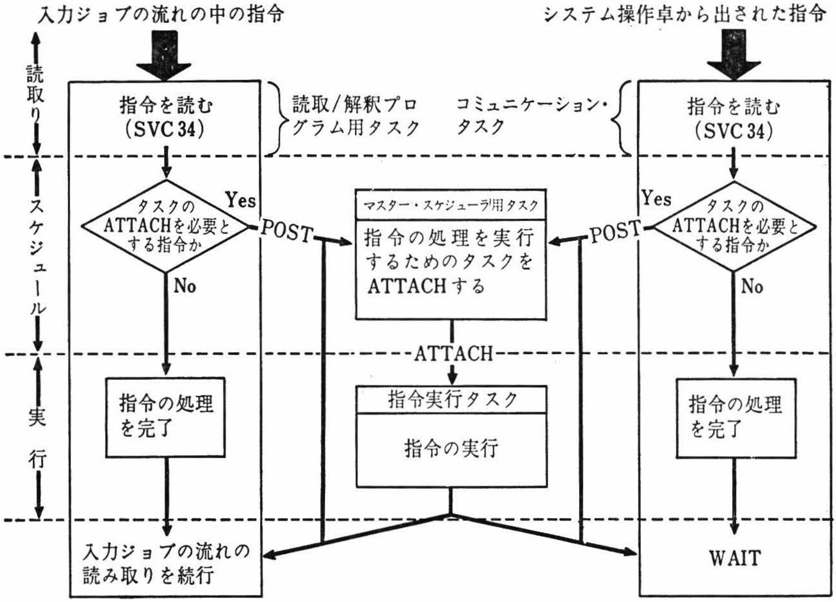


図 7-31　指令 (command) の処理ステップ

解釈プログラム用タスクで遂行)

- 指令のスケジューリング（マスター・スケジューラー用タスクで遂行）
- 指令の実行（指令実行タスクで遂行）

これら3つの局面は，かつこの中で示したような種々のシステム・タスクのもとで遂行される．ある種の指令は，1つのシステム・タスクの一部で処理されるが，大半の指令の処理では複数個のシステム・タスクがかかわりをもつ．指令を処理するステップとシステム・タスクの関係は，**図 7-31**で示している．以下で，それぞれのステップとシステム・タスクの機能について説明する．

① 指令の読取り

オペレーターの指令は，コンソールか入力ジョブの流れからシステムに入力される．コンソールから入力された指令を読み取る仕事は，コンソール・コミュニケーション・タスクのもとで実行されるルーチンが受持つ．一方，入力ジ

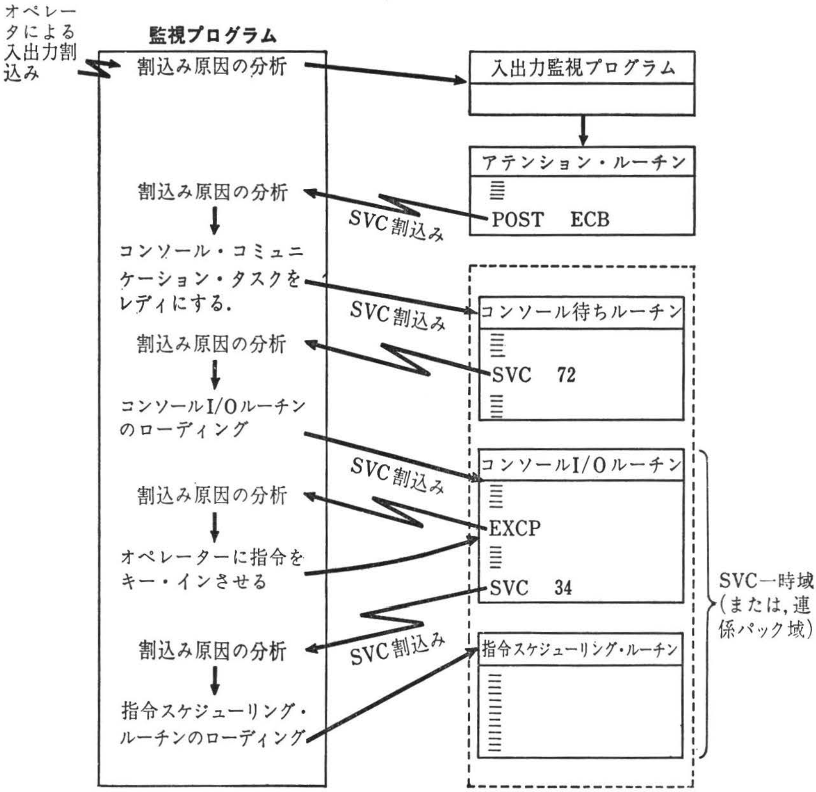


図 7-32　コンソール指令の流れ

ョブの流れから入力された指令を読むのは，入力ジョブの流れとかかわり合う読取りタスクのもとで実行されるルーチンである．

コンソール・コミュニケーション・タスク (console communication task) はシステム生成時に作られる．このタスクは，独自の領域を占有するのではなく，マスター・スケジューラー用タスクに割り当てられた領域を共用する．

ここで，コンソールを通じて指令を出した場合の制御権の流れをみておこう．オペレーターが指令を入力するのに先立って“アテンション”ボタンを押すか，装置をレデイにすると入出力割込みが生じる．図 7-32の左上を参照していただ

きたい．入出力割込みによってまず監視プログラムに制御権が渡され，割込み原因が調べられる．入出力割込みであるから，制御権は入出力監視プログラムに移される．入出力監視プログラムがこの割込みの原因をつきとめたあと，**アテンション・ルーチン**（attention routine）に制御権を渡す．このルーチンでは，コンソール・コミュニケーション・タスクで待ち状態にある**事象制御ブロック**（Event Control Block：略して ECB）に対して POST マクロ命令を出す．これによって，監視プログラムはコンソール・コミュニケーション・タスクをレディにし，このタスクのルーチンに CPU の制御権を移す．

コンソール・コミュニケーション・タスクが行う指令の読取りは，実際には**コンソール待ちルーチン**（console wait routine）と**コンソール I/O ルーチン**（console I/O routine）によって遂行される．コンソール待ちルーチンは中核プログラムの一部をなしており，コミュニケーション・タスクにおける制御的機能を果すルーチンである．このルーチンが，コミュニケーション・タスクのためにまず制御権を受け取り，このタスクが完了したときにタスクを待ち状態に置くための WAIT マクロ命令を出す．さらに，コンソール待ちルーチンは，監視プログラムに制御権を委譲し，監視プログラムがコンソール I/Oルーチンをローディングしたうえで，そのルーチンに制御権を渡す．

コンソール I/Oルーチンには，コンソールとして使用される入出力装置に対応したサポート・ルーチンが用意されている．各々のルーチンは基本的には同じ機能を果すのであるが，装置の種類によって若干異った作動が要求されることから装置ごとに異ったコンソール I/Oルーチンがある．ともあれ，このルーチンで，指令を受け入れるための記憶域を確保したあと，指令が転送できるようにするために EXCP マクロ命令を出す．EXCP マクロ命令にもとづいて，コンソールから指令がキー・インされてしまうと，指令スケジューリング・ルーチンが別のシステム・タスクを必要とする指令をスケジュールし，指令の実行に入ってゆくのである．

一方，入力ジョブの流れから指令を読み取る場合には，入力ジョブの流れの処理を担当する**読取／解釈プログラム用タスク**（reader/interpreter task）が指令の読取りと分析を行う（図 7-33参照）．このタスクの**解釈制御ルーチン**（in-

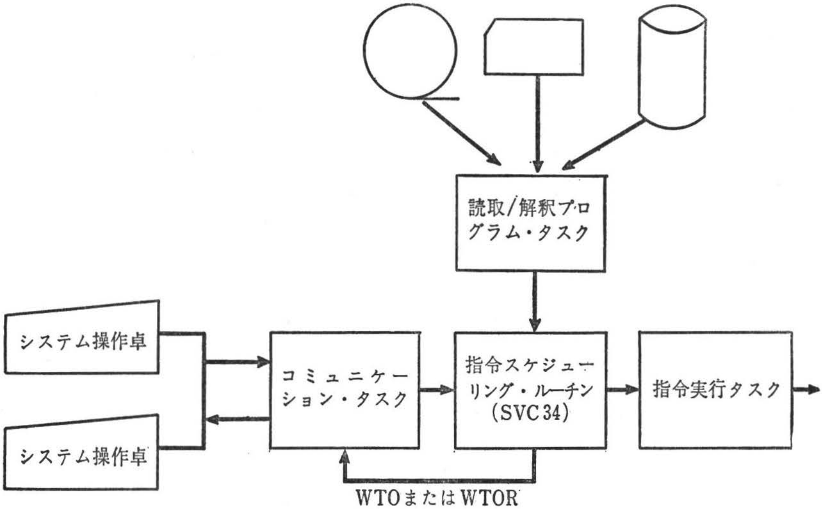


図 7-33　指令処理のためのタスクの相互関係

tepeter control routine）が指令をみつけだすと，自動的に指令スケジューリング・ルーチンを呼び出し，制御権を渡す．指令スケジューリング・ルーチンは，コンソール・コミュニケーション・タスクの場合と全く同じ方法で指令を分析し，スケジュールを行う．読取／解釈プログラム用タスクのための処理が完了すると，CPU 制御権を再び解釈制御ルーチンに戻し入力ジョブの流れの読取りが続行できるようにする．

② 指令のスケジューリング

指令のスケジューリングとは，指令を一時的に貯え，その指令の処理を続けるための別のシステム・タスクを ATTACH することである（図7-31参照）．このために機能するのが指令スケジューリング・ルーチンである．指令スケジューリング・ルーチンは，指令がコンソールから入力された場合にはコンソール・コミュニケーション・タスクのもとで働き，指令が入力ジョブの流れから入力された場合には読取／解釈プログラム用タスクのもとで働く．

入力された指令を実行するにあたって特別なシステム・タスクを必要としない時には，指令スケジューリング・ルーチンが指令を処理し，元の呼び出しルーチンに制御権を戻す．コンソール・コミュニケーション・タスクの場合だと，コンソール I/Oルーチンが制御権を受け取り，そのあとでコンソール待ちルーチンに制御権を渡す．コンソール待ちルーチンは，コミュニケーション・タスクを待ち状態に置く．一方，読取／解釈プログラム用タスクの場合だと，前述したように制御権を解釈制御ルーチンに戻し，入力ジョブの読み取りが続行されるようにする（図7-31参照）．

しかし，指令の処理のためには余分のシステム・タスクが要求されるのが普通である．指令を実行するためのタスクがシステム内に存在しないときには，指令スケジューリング・ルーチンで**指令スケジューリング制御ブロック**（Command Scheduling Control Block：略して CSCB）を作りあげ，CSCB 待行列の中に入れてから，マスター・スケジューラー・タスタに知らせる．そこで，マスター・スケジューラー・タスクはその指令を実行するに必要なシステム・タスクを ATTACH する（図7-31参照）．もしも，指令を実行するのに必要なシステム・タスクがすでにシステム内に存在している場合には，CSCB 内の ECB にこの旨が知らされる．したがって，この場合には，新たにシステム・タスクを ATTACH することはない．

③ 指令の実行

すでに説明してきたように，入力された指令はつぎの３つのカテゴリーのいずれかのシステム・タスクのもとで実行される：

- 指令が読み取られたと全く同じタスク，つまりコンソール・コミュニケーション・タスクまたは読取／解釈プログラム用タスクのもとでの実行．
- すでにシステム内に存在するタスクで，指令の実行以外の機能をも持ち合わせているようなタスクのもとで実行．
- 指令を実行するために特別に作りあげられたシステム・タスクのもとでの実行．

指令を読み取ったタスクのもとで実行できる指令はあまり多くはない．この種の指令を指令スケジューリング・ルーチンが見つけると，すべての必要な処理を行ったあとですぐに制御権を呼び出しルーチンに戻してしまう．

ある種のタスクは，2種類以上の指令を実行するための機能がもたされている．たとえば，MODIFY 指令を処理するためのルーチンは，START 指令に呼応して作られたタスクの制御下で働くことができるようになっている．この種のものが2番目のカテゴリーに入る．

なお，ジョブ処理用タスク，つまり読取／解釈プログラム用システム・タスク，開始／終止プログラム用システム・タスクおよび出力書出しプログラム用システム・タスクは，指令の実行のために特別に作られる．つまり，START 指令が出されるつど，新しいジョブ処理用タスクが要求された機能を果すために作られるのである．

4) ジョブ処理の内部構造

ジョブ処理は，つぎの3種類のシステム・タスクによって遂行される：

- **読取／解釈プログラム用システム・タスク：** 入力ジョブの流れの読取りと制御ステートメントの解釈をコントロールする機能
- **開始／終止プログラム用システム・タスク：** すでに読取りと解釈が済んだジョブ・ステップの開始と終了をコントロールする機能
- **出力書出しプログラム用タスク：** システム・メッセージおよびユーザーのデータ・セットを直接アクセス・ボリュームから印刷装置，穿孔装置あるいは磁気テープのような外部記憶媒体に転送する仕事をコントロールする機能

これら3種類のタスクはいずれも START 指令によって作り出される．この3つのジョブ処理の機能を果すルーチンは，すでに説明してきたようにジョブ・スケジューラーという言葉で総称される．ともあれ，START 指令が出されると，マスター・スケジューラー・タスクのルーチンが “START 指令” タスクを ATTACH する．この START 指令タスクの特別なルーチンである**シ**

ステム・タスク制御ルーチン（system task control routine）が開始すべきタスクを判定し，読取／解釈プログラム用タスク，開始／終止プログラム用タスクないしは出力書出しプログラム用タスクのうちのいずれかをATTACHする．一方，これらのタスクの実行が終ると，START指令タスクのルーチンが終了したタスクをDETACHし，ある種の事後処理を行ってくれる．そのあとで，システム・タスク制御ルーチンが制御権を監視プログラムに渡し，監視プログラムがはじめてSTART指令タスクを終了させる仕組みになっている．

ジョブ処理用タスクは，いずれも1個以上用意することが許されている．たとえば，複数個の入力ジョブの流れに対してそれぞれ“START READER”指令を出すことによって，数個の入力装置からジョブの流れを読み取らせてもよい．各々の指令で装置ごとの読取／解釈プログラム用タスクが作り出されるのである．また，各装置に対応する“START WRITER”指令を出すことによって，システム・メッセージやデータ・セットを複数個の出力装置に書き出すこともできる．同様に，“START INITIATOR”指令によって複数個の開始／終止プログラム用タスクを作りあげることも可能である．これらは，主記憶装置の利用可能性と処理効率を勘案したうえで決定すべき事柄である．

ともあれ，ここでは読取／解釈プログラム用タスク，開始／終止プログラム用タスクおよび出力書出しプログラム用タスクの処理構造について概観しておくことにする．

① 読取／解釈プログラム用タスク処理構造

読取／解釈プログラム用システム・タスクの構造を示したのが**図7-34**である．この図をもとにその流れを説明してゆこう．

マスター・スケジューラーがSTART指令を受け取ると，START指令タスクをATTACHし，システム・タスク制御ルーチンに制御権を渡す（①）．システム・タスク制御ルーチンは，読取／解釈プログラム用タスクを作り，それを開始させる．読取／解釈プログラム用タスクが制御権を受け取ると，ある種の初期値化の仕事を行ってから(②)，入力ジョブの流れとプロシージャー・ライブラリーのデータ・セットをOPENし，かつ両者の論理的結合を計る(③)．

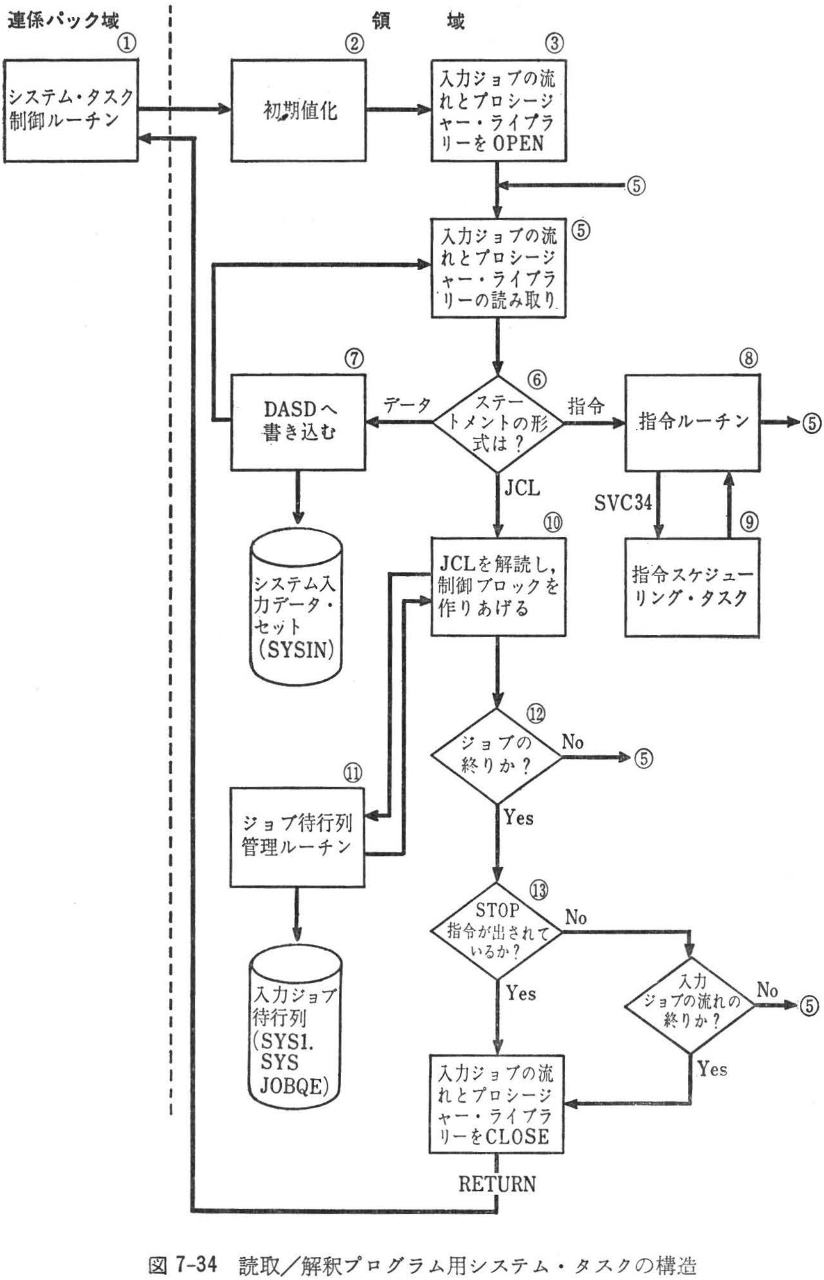


図 7-34　読取／解釈プログラム用システム・タスクの構造

この結果，双方のデータ・セットのレコードが処理できるようになる．このあと，入力ジョブの流れと，必要に応じてプロシージャー・ライブラリーからステートメントを読み取る(⑤)．読み取ったステートメントがどの種のタイプかを決定する(⑥)．入力ジョブの流れに指令があると指令ルーチンに移り，その処理を行う（⑧と⑨)．これに関しては，指令処理の個所で詳述した通りである．指令の処理が終ると，制御権はつぎのステートメントの読み取りに移される．入力ジョブの流れの中にデータがある場合だと，入力データ・セットとして DASD 上に転送される（⑦)．１レコード転送するごとに⑤にもどり，つぎのレコードを読み取り転送する．読み取ったステートメントがジョブ制御ステートメントだと，それを解読し一定形式の制御ブロックに仕立てあげる（⑩)．そのあとで，**ジョブ待行列管理ルーチン**（job queue manager）に制御権を渡す．ジョブ待行列管理ルーチンは，入力ジョブ待行列の中に適当なスペースを確保して制御テーブルを収容すると同時に制御テーブル間のつながりを示すための連鎖用ポインターを挿入してから，再び⑩にもどる．１つのジョブ制御ステートメントの処理が終るつど，ジョブの終りのステートメントに到達したかどうか確める(⑫)．１つのジョブの読み取りと処理が完了していないときは，⑤にもどり，次のステートメントを読み同様の処理手順を踏む．ジョブの読取りと解釈が終了すると，読取／解釈プログラム用タスクを終了させるための指令が出されていないかどうか調べる（⑬)．STOP 指令が出されていない場合には，入力ジョブの流れが終りになっているかどうか確める．終りでないなら，当然のことながら⑤にもどって，つぎのジョブの読み取りを開始する．一方，STOP 指令が出されていたり，入力ジョブの流れの終りに達している場合だと，入力ジョブの流れとプロシージャー・ライブラリーの双方を CLOSE し，論理的結合を切り離したうえで，システム・タスク制御ルーチンに制御権を戻す．この結果，読取／解釈プログラム用タスクが終了したことがシステムに知らされることになる．再びこのタスクを開始させるためには，START 指令を出さなければならない．

② 開始／終止プログラム用タスクの処理構造

開始／終止プログラム用システム・タスクの処理構造は，**図7-35**に示してある．この図にもとづいて処理の流れをみてゆこう．

開始／終止プログラム用タスクは，まずジョブ待行列ルーチンを呼び出し，入力待行列の中から最も優先順位の高いジョブを取り出させる．ひとたびジョブが選択されると，そのジョブを構成するジョブ・ステップの開始，実行および終了は，開始／終止プログラム用タスクの制御のもとで順番に遂行されてゆく．したがって，複数個の開始／終止プログラム用タスクが同時に異ったジョブのステップの開始，実行，終了をそれぞれ制御している場合には，**多重ジョブ処理**（multi-jobbing）が実現できるのである．

多重ジョブ処理のもとでは，同時併行的に処理される2つ以上のジョブで1つのデータ・セットを利用する可能性がある．したがって，ジョブの最初のステップを開始するまえに，開始プログラムが監視プログラムの中の ENQ/DEQ ルーチンを呼び出して，該当ジョブで必要となるデータ・セットが同時に他のジョブでも使用されており，かつジョブ間で同時使用ができない性質のデータ・セットかどうかを決定させる．もし，そのジョブを構成するいずれかのジョブ・ステップで必要となるデータ・セットが同時使用できない場合には，そのジョブの開始は，データ・セットが利用可能になるまで延期される．また，そのジョブの中で使用するデータ・セットのうち，他のジョブに同時使用を許さないものがある場合には，そのジョブの実行中そのデータ・セットを他のジョブからアクセスできないようにするために，データ・セットを ENQ する．なお，すべてのジョブでデータ・セットの共有ができる旨の指示がある場合には，そのデータ・セットの同時使用が許されることはいうまでもない．

そのジョブが使用するデータ・セットがすべて利用可能だと，最初のジョブ・ステップに必要となる主記憶域が要求される．主記憶域，つまり領域が確保されると，開始／終止プログラムは該当ステップを選択し，入力データ・セットの所在のつきとめと必要な入出力装置の割り当てを行う．

ジョブ・ステップの開始準備の最後の仕事が，該当ジョブ・ステップのタスクを生成することである．ATTACH マクロ命令を出すことによって，CPU の制御権を ATTACH SVC ルーチンに移す．このルーチンが TCB を作りあげ，

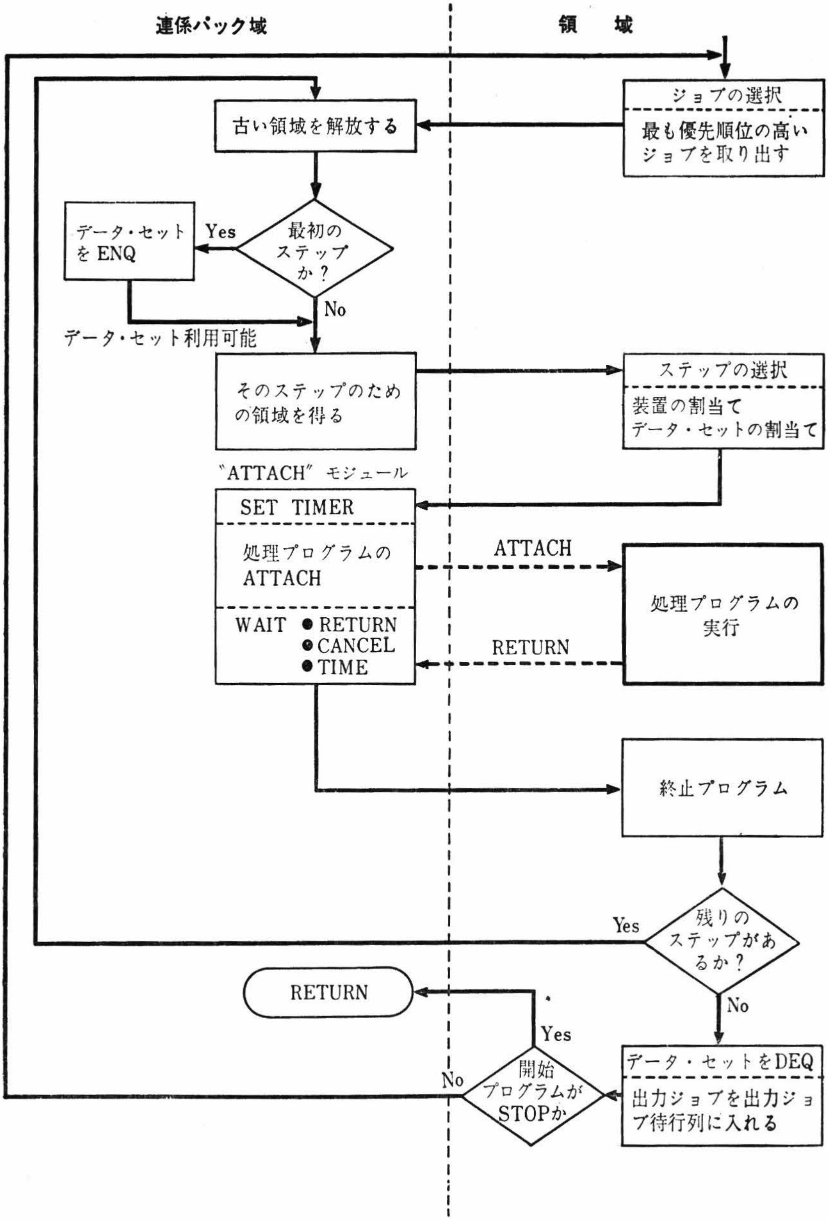


図 7-35　開始／終止プログラム用システム・タスクの構造

制御プログラムでジョブ・ステップの処理を制御してゆくことができるようにする．したがって，この TCB のことを一般にジョブ・ステップ用 TCB という．TCB は，ジョブの優先順位にもとづいて，TCB 待行列の中に置かれる．これが終ると，制御権は，処理プログラムに渡される．

ジョブ・ステップが完了するか，指定された時間が経過するか，異常終止するか，ないしはオペレーターによってジョブが取り消されるかのいづれかで，ジョブ・ステップは終了する．これらのどれかの条件によって，監視プログラムはそのジョブ・ステップの開始を制御した開始／終止プログラム用タスクをレディの状態にする．このレディになったタスクの制御にもとづいて，ジョブ・ステップを終止させるための処理が行われる．終止プログラムがそのステップに割り当てられた領域に呼び込まれ，そのジョブ・ステップで作りあげたり使用したデータ・セットの事後処理を行い，そのステップに割り当てられた入出力装置を解放する．さらに，ジョブ・ステップ用 TCB を DETACH マクロ命令を通じてシステムから切り離してしまう．そのあとで，後続するジョブ・ステップがあるかどうか確かめる．後続ステップがある場合には，前のステップで使用した領域を解放したうえで，これまで説明した手順がとられる．ジョブの終りの場合だと，ENQ したデータ・セットを解放し，そのジョブで作り出された出力データ・セットに対する出力ジョブの情報を出力待行列に入れる．そのあとで，再び後続のジョブの選択に入る．入力ジョブ待行列にジョブがない場合だと，マスター・スケジューラーの待ちルーチンに制御権が移されてしまう．

③ 出力書出しプログラム用タスクの処理構造

図 7-36に示したのが，出力書出しプログラム用システム・タスクの処理構造である．出力書出しプログラム用タスクは，指定されたクラスのすべてのシステム・メッセージおよび SYSOUT データ・セットを DASD 上からシステム出力装置（SYSOUT）に書き出す仕事を制御する．

出力書出しプログラム用タスクは，処理すべきメッセージおよびデータ・セットのクラスを指示した“START WRITER”指令の結果として作り出される．

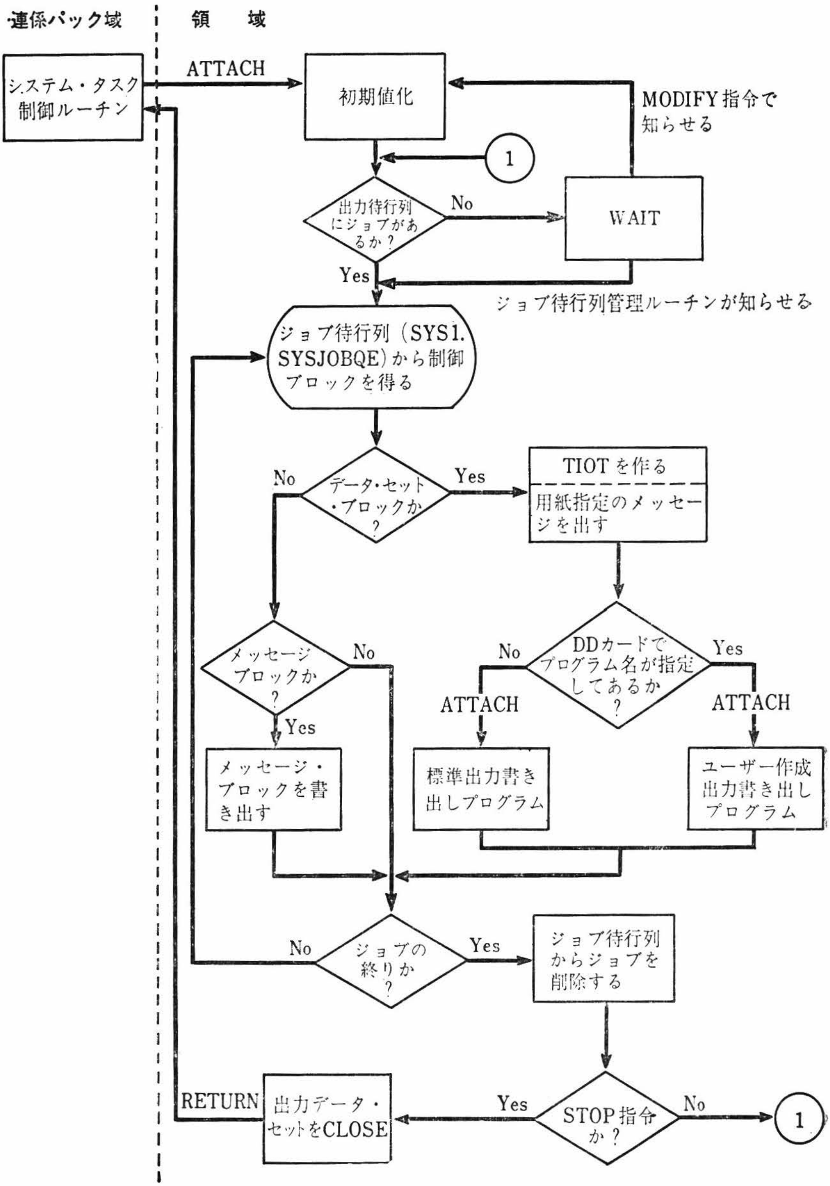


図 7-36　出力書出しプログラム用システム・タスクの構造

START 指令タスクのシステム・タスク制御ルーチンがこのタスクを ATTACH するのである. 出力書出しプログラム用タスクの制御のもとに一連の初期値化作業を行う. 出力待行列にジョブがある場合には, ジョブ待行列管理ルーチンにどのメッセージ・クラスを書出すのかを知らせる. ジョブ待行列管理ルーチンは, 該当クラスの出力ジョブ待行列の中から最も優先順位の高い出力ジョブの制御ブロックを取り出し, 出力書出しルーチンに渡す. 出力書出しルーチンでは, データ・セット・ブロックかメッセージ・ブロックかに応じてそれぞれにふさわしい書出しを行う. すべてのメッセージとデータ・セットの書出しが終ると, つぎの出力ジョブをジョブ待行列管理ルーチンに要求し, 再び同様の処理手順を続ける.

該当クラスのすべてのデータ・セットとメッセージの書き出しが行われると, 該当クラスの追加データが処理可能になるか, "STOP WRITER" 指令が出されるまで待ち状態に入る. 出力ジョブ待行列に出力ジョブが登録されると, 出力書出プログラム用タスクは再びレディの状態に戻り, 処理を再開する. 一方, STOP 指令が出された場合には, 出力データ・セットを CLOSE したあとでシステム・タスク制御ルーチンに制御権をもどしてしまう. この結果, 出力書出しプログラム用タスクが終了したことがシステムに知らされる.

四. ジョブ制御ステートメントの構造

これまで, ジョブ管理に関して, それが出現する歴史的背景を概観したうえで, ジョブ管理の機能構造をジョブ・スケジューラーとマスター・スケジューに大別して説明してきた. さらに, ジョブ管理のシステム構造として, 順次式ジョブ・スケジューリング・システムと優先式ジョブ・スケジューリング・システムをとりあげて検討してきた. いずれの場合にも, ジョブを連続的に実行するという使命を果すためには, 各ジョブを規定する情報が外部から与えられる必要がある. ジョブに関する情報を一定形式で集約して与えるために使われるのが**ジョブ制御言語** (Job Control Language : 略して JCL) であり, その結果ジョブ・スケジューラーへの入力となる**入力ジョブの流れ** (input job stre-

am）が作りあげられる.

ここでは，ジョブ制御言語を用いて作成するジョブ制御ステートメントについて説明してゆく．ジョブ制御ステートメントの一般的書き方を具体例にもとづいて紹介したうえで，各ステートメントの指定内容を説明してゆく．これによって，ジョブ制御ステートメントの機能構造を明らかにすると同時にジョブ・スケジューラーの果す役割りをより明確にしてゆきたい.

1.　ジョブ制御ステートメントの作成規則と基本的機能

ジョブ・スケジューラーの目的は，一口に言えば，システムの生産性を最大限に高めてゆくことにある．このためシステム資源を有効に管理し，ジョブを連続的に処理してゆくことによって，この目的を達成しようとしている．したがって，どのようなジョブをどのような順序で行なおうとしているのか，各ジョブ・ステップでどのようなシステム資源を必要としているのかなどをジョブ・スケジューラーに知らせる必要がある．この種の情報を提供するために使われるのがジョブ制御言語を用いて作成される**ジョブ制御ステートメント**（job control statement）である.

このジョブ制御ステートメントの作成規則を知ってもらうために，まずは簡単なサンプルをとりあげて解説を加えてゆくことにしよう.

1）　ジョブ制御ステートメントの作成規則

図7-37で極めて簡単なジョブの内容を図示している．給与データ・セットを読んで，給与一覧表データ・セットを作成するジョブである．いずれのデータ・セットも順次編成である．このジョブを処理するのに使用されるプログラムには，KYUYO という名前がつけられて，あらかじめ SYS1. LINKLIB に登録されているものとする．なお，入力データ・セットである給与データ・セットは，入力ジョブの流れの中に置くものとする．また，KYUYO プログラムの中では，給与データ・セットに対して INDD というデータ定義名（DD 名）を，給与一覧表データ・セットに対しては OUTDD というデータ定義名を付けているものとしている（アセンブラー言語では DCB マクロの中の DDNAME パ

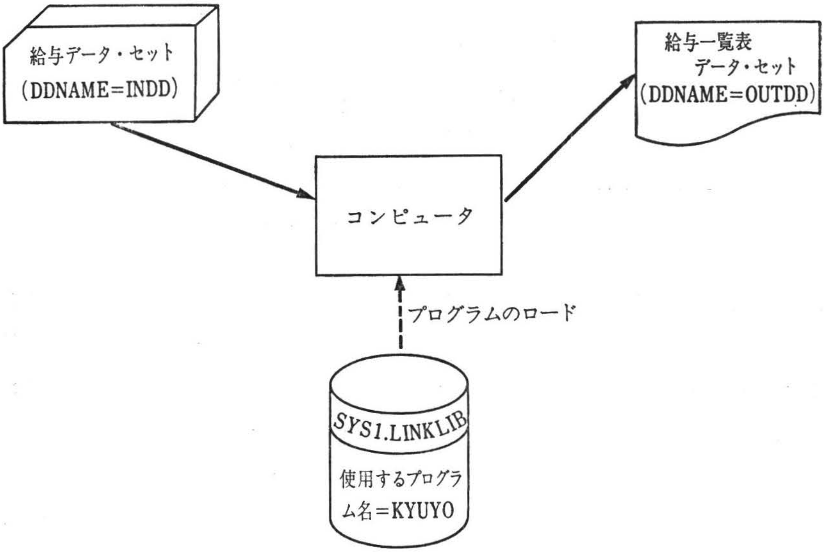


図 7-37　給与一覧表データ・セットを作成するジョブ

ラメータでデータ定義名を与える．第6章「データ・セットの編成法とアクセス方式」を参照のこと)．

このような条件のジョブを実行するためのジョブ制御ステートメントは，図7-38で示したようになる．第37図と対比して，まず相互関係を把握して欲しい．

最初にジョブの始まりを示すために JOB ステートメントがくる．JOB ステートメントの次にあるEXECステートメントが，ジョブ・ステップの始まりを

```
//JOB1       JOB     MSGLEVEL=2
//STEP1      EXEC    PGM=KYUYO
//INDD       DD      *
              :      入力データ（給与データ・セット）
/*
//OUTDD      DD      SYSOUT=A
//
```

図 7-38　「図7—37」のジョブのためのジョブ制御ステートメント

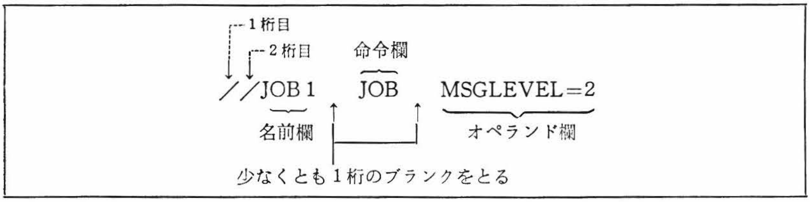


図 7-39　JOB ステートメントの形式

示す．このジョブは，1つのジョブ・ステップだけから成り立っているので1個の EXEC ステートメントだけで事足りる．このジョブ・ステップでは2種類のデータ・セットを用いているので，それぞれに対応してDDステートメントが用意される．最初のDDステートメントでは，入力ジョブの流れにある給与データ・セットについて定義している．このDDステートメントに続けて給与データ・セットを置き，給与データ・セットの終りは区切ステートメント（/*）で示している．一方，出力データ・セットである給与一覧表データ・セットは，もう一個のDDステートメントで定義している．ジョブの終りを示すのには，空白ステートメント（終了ステートメントともいう）を用いている．以下で各ステートメントの基本的な機能とそれぞれの作成規則について説明してゆこう．

JOBステートメントは，ジョブの始まりを示すと同時に2個以上のジョブが連続している場合には，先行ジョブの制御ステートメントが終ったことを示すために用いる．

図 7-39 で JOB ステートメントの記入形式を示している．1桁目と2桁目に／を書き，3桁目から始まる名前欄にジョブ名を記入する．この例では，ジョブ名として JOB1 を使っている．ジョブ名は，8桁以内の英数字を使って任意に指定できる．ただし，ジョブ名の最初1けたは英字にする必要がある．最初の文字を数字とすることは許されない．ジョブ名のあとには1桁以上のブランクをとったうえでJOBと記入する．命令欄がJOBであることによって，JOBステートメントであることを示す．命令欄のあとに1桁以上のブランクをあけて，オペランドを記入する．オペランドでは，さまざまな情報を与えることが

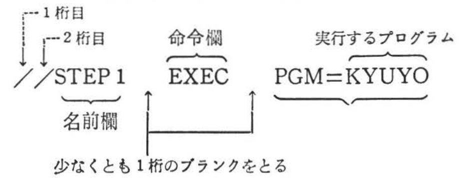


図 7-40　EXEC ステートメントの形式

できるが，これについては，次節で詳述する．なお，この例で示されている MSGLEVEL=2 は，ジョブの終りに書き出される出力リストに，ジョブ制御ステートメントも書き出すことを指示している．オペランドを記入したあと1桁以上のブランクをとると自由にコメントを与えることもできる．

EXEC ステートメントは，ジョブ・ステップの始まりを示すと共に，そのジョブ・ステップで実行するプログラム名を指示するのに用いる．

EXEC ステートメントの記入形式を示したのが，**図 7-40**である．EXEC ステートメントの形式は，JOBステートメントと同じく1桁目と2桁目に／を記入し，名前欄が続く．名前欄のあとには，1桁以上のブランクをとって命令欄が続く．このあとに1桁以上のブランクをとって，オペランド欄を書く．

EXEC ステートメントの名前欄には，**ジョブ・ステップ名**（job step name）を記入する．ジョブ・ステップ名の指定方法は，ジョブ名の場合と同じである．つまり，8桁以内の英数字を使ってユーザーが任意に指定できる．当然のことながら，ジョブ・ステップ名の最初の文字も英字である．ジョブ・ステップ名は必要がない時には省略してもかまわない．命令欄には EXEC を書き，EXEC ステートメントであることを表示する．そのあと1桁以上のブランクをとって，そのステップで使用するプログラム名をオペランドとして与える．プログラム名は，プログラム・ライブラリーにプログラムを登録するさいにつけた名前である．EXEC ステートメントのオペランドでは，プログラム名以外の種々のことが指定できるが，これについては後述する．なお，オペランドのあとに1桁以上のブランクをとってコメントを記入することが許される．

EXEC ステートメントのあとには，そのジョブ・ステップで使用するデータ

・セットを定義するDDステートメントを続ける．このステートメントでは，データ・セットの諸特性を指定し，ジョブ・スケジューラーに入出力資源の割り振りを要求する．

図7-41に，DDステートメントの形式が示されている．この例は，入力ジョブの流れの中にデータをおいて入力する場合であり，区切りステートメントもあわせて示してある．

DDステートメントの形式も，JOB, EXECステートメントと同じく，1桁目と2桁目は／で，次の欄は名前欄，このあとに命令欄とオペランド欄が続く．名前欄と命令欄の間，命令欄とオペランド欄の間には1桁以上のブランクが必要である．名前欄に記入するのは，処理プログラムの中でデータ・セットに付けたデータ定義名である．

入力ジョブの流れの中にデータを置くときは，データの始めとデータの終りを示す情報が必要である．データの始めは，DDステートメントのオペランドに＊を書いて示し，一方データの終りは区切りステートメントで示す．区切りステートメントでは，1桁目に／，2桁目に＊を書く．

図7-38の2番目のDDステートメントのオペランドはSYSOUT＝Aである．これは，出力データ・セットをクラスAとしてDASD上のシステム出力データ域に書き出すことを指示している．DASD上のシステム出力データ域に書き出した内容は，あとで，クラスAで働いている出力書出しプログラムによって，印刷装置に書き出される．これに関しては，前節の出力書出しプログラムの機能構造に関する説明を参照されたい．なお，DDステートメントのオペラ

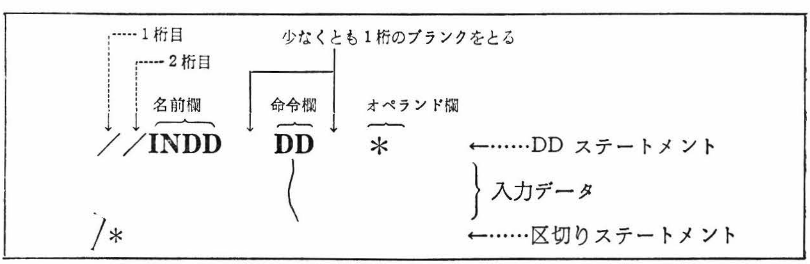


図7-41　DDステートメントと区切りステートメントの形式

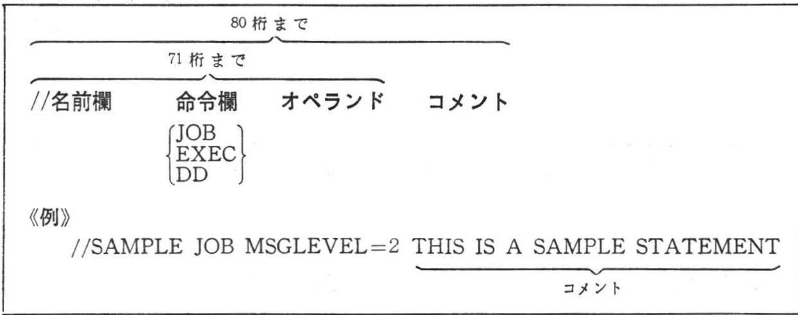


図 7-42 ジョブ制御ステートメントの一般形式

ンドとしては実にさまざまなパラメータが指定できる．これに関しても別項目を設けて詳述する．また，JOB, EXEC ステートメントの場合と同様の規則でコメントを記入することも許されている．

ジョブの最後の制御ステートメントは，1桁目と2桁目の／だけからなるステートメントで，**空白ステートメント**（null statement）または終了ステートメントといわれる．空白ステートメントは，いま読み取ったジョブの定義を入力ジョブ待行列に入れるようにジョブ・スケジューラーに伝達する役目をしている．したがって，ジョブの終りに入れる．ただし，この空白ステートメントがなくても，次の JOB ステートメントがきたとき，あるいは入力ジョブの流れの終りをジョブ・スケジューラーが知ることによって，自動的にジョブの定義を入力ジョブ待行列に入れてくれるので，必ずしも空白ステートメントは必要ではない．しかし，空白ステートメントを入れることによって，次のジョブ・ステートメントの誤りによる影響を防ぐことができるので，入れておいた方がよいといえよう．

以上の説明を念頭に図7-38をもう一度追跡していただきたい．**図 7-42**でこれまで説明してきたジョブ制御ステートメントの一般形式を示しておいた．これをもとに，これまでの説明をまとめると同時に若干新たなことをつけ加えておこう．通常のジョブ・ステートメントでは，1桁目と2衍目に／を書き，そのあとに名前欄，命令欄，オペランド欄が続く（ただし，区切りステートメントと空白ステートメントは，名前欄，命令欄，オペランド欄はない）．これらの

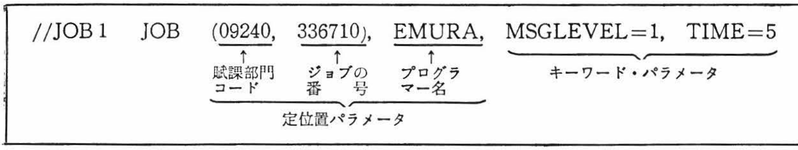


図 7-43　定位置パラメータとキーワード・パラメータ

あとに1桁以上のブランクをあけてコメントを記入できる．コメントは，システムに対して何らの指示的役割りも果さないが，文書化のために必要な情報を示しておくと，あとで便利である．ともあれ，コメントは80桁まで記入できるが，他の情報は71桁目までしか書けない．71桁を越える場合だと，継続の指示をして，2行以上にまたがって書くことができる．継続を示すためには，最初の行の72桁目にブランク以外の文字を記入し，つぎの行の1桁目と2桁目に／を入れる．継続行の内容は4桁目から16桁目の間から書き始めなければならない．

ジョブ制御ステートメントのオペランド欄に記入するパラメータは，ジョブ・スケジューラーがジョブないしはジョブ・ステップの実行を規制したり，データ・セットの取出しや後始末をしたり，入出力装置の割当てやその解除をしたり，あるいはオペレーターとのコミュニケーションを行ったりするために使われる重要な情報である．ところで，パラメータを指定するには，**定位置パラメータ**（positional parameter）と**キーワード・パラメータ**（keyword parameter）と呼ばれる2種類の方法がある（**図 7-43**参照）．特殊なパラメータは，どちらかの方法を選択して指定できるが，ほとんどのパラメータはいずれかの方法で指定することがあらかじめ定められている．

図7-43で示しているように，定位置パラメータは，指示された順序で，かつキーワード・パラメータより前に書かなければならない．途中の定位置パラメータを省略する場合にはカンマ（，）でその旨を指示する必要があることはいうまでもない．一方，キーワード・パラメータは定位置パラメータのあとに続けるが，全ての定位置パラメータが省略されている場合には，カンマを入れることなくオペランドの最初から記入できる．また，キーワード・パラメータはどんな順序で書いてもよい．定位置パラメータのように特別な順序はない．

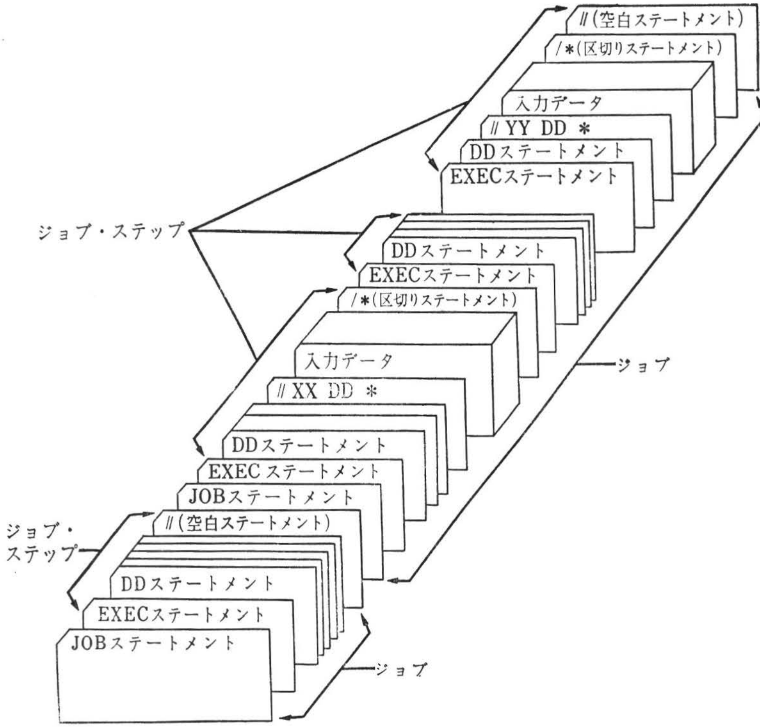


図 7-44　入力ジョブの流れの例

2) 入力ジョブの流れの構成

入力ジョブの流れは，ジョブまたはジョブ・ステップを定義するジョブ制御ステートメントから構成される．ただし，図7-38の例でみたように入力ジョブの流れの中にデータそのものを入れることは許される．

図 7-44に示しているように，ジョブは１つのジョブ・ステップから構成されることもあるし，複数個のジョブ・ステップから構成されることもある．ジョブとジョブ・ステップの相互関係については，第２章「オペレーティング・システムの概念的構造」の「三．システムの構成要素と相互関連性」の中で具体的に説明しておいたので参照されたい．この図において，２番目のジョブの最

初のジョブ・ステップと3番目のジョブ・ステップでは，データも入力ジョブの流れの一部としている．これらのデータは，ジョブ・スケジューラーの読取/解釈プログラムが読みとって DASD 上の入力データ・セット域に一時的に貯えてくれる性格のものである．なお，入力ジョブの流れは，必ずしも穿孔カード読取装置から読取らせる必要はない．磁気テープ装置，磁気ディスク記憶装置，データ通信端末装置または映像表示装置などから入力してもよい．これらの場合には，ジョブの流れを読取るための特別な読取/解釈プログラムを必要とするだけである．

入力ジョブの流れがいくつあっても，それぞれの読取/解釈プログラムが読取ったジョブ制御ステートメントは一定の制御テーブルに集約されたうえで，1個の入力ジョブ待行列（SYS 1. SYSJOBQE データ・セット）に記録されてしまう．一方，入力ジョブの流れにあるデータは先述したように DASD 上に一時的データ・セットとして貯えられるのが普通である．これらの一時的データ・セットについては，システムが適当なデータ・セット名を割り振ってくれる．これに関しては，図7-20を再度参照しておいて欲しい．

3) ジョブ制御ステートメントの機能の要約

572頁から各ジョブ制御ステートメントをとりあげて，その具体的機能と指定方法を詳述してゆく．ここでは，その橋渡しとするために各ジョブ制御ステートメントの機能をまとめておくことにする：

① **JOB ステートメント**： ジョブの始まりを示す．いくつかのジョブが連続している場合には，先行ジョブの終りをも指示する．

このステートメントで与えることのできる主な情報としてはつぎのものがある：

- ユーザーが組込んだ**会計ルーチン**（accounting routine）のための各種会計用情報
- ジョブの終了条件
- ジョブ・クラスの割当て

- ジョブの優先順位の割当て
- ジョブ・スケジューラーが出すメッセージの印刷指示
- ジョブの実行の制限時間
- ジョブに割当てる記憶域（領域）の大きさ

② **EXECステートメント：** ジョブ・ステップの始まりを示す．

このステートメントで与えることのできる主要な情報としてはつぎのようなものがある：

- そのジョブ・ステップで実行すべきプログラムの名前または呼び出すべきカタログ式プロシージャーの名前
- 各種会計用情報
- ジョブ・ステップの優先順位の割当て
- ジョブ・ステップに必要な領域の大きさ
- ジョブ・ステップの実行の制限時間
- ジョブ・ステップの実行あるいはバイパスの条件
- 処理プログラムに渡すパラメータ情報

③ **DDステートメント：** データ・セットに関する記述を行い，入出力資源の割り当てを要求する．

このステートメントで与えることのできる主な情報としては，つぎのようなものがある：

- データ・セット名
- 使用する装置，およびボリュームに関する情報
- データ・セット・ラベルに関する情報
- データ・セットに関する諸属性

④ **区切りステートメント：** 入力ジョブの流れの中のデータと後続するジョブ制御ステートメントとを分離するのに使用する．順次式スケジューリング・システムでは必ず必要であるが，優先式ジョブ・スケジューリング・システムでは特別な場合のみ必要である．

⑤ **空白ステートメント：** ジョブの終りを示すのに使われる．必ずしも必要ではない．

⑥ **PROC ステートメント：** プロシージャーの最初のステートメントとして使われる．**インライン・プロシージャー**（in-line procedure）ではプロシージャーの最初を示す．

⑦ **PEND ステートメント：** インライン・プロシージャーの終りを示す．

⑧ **指令ステートメント：** 入力ジョブの流れを通じて，オペレーターの指令を入力する目的で使う．

2. JOB ステートメントでの主要な指定内容

ジョブ・ステートメントの指定内容には，必ず指定する必要のあるものと，任意選択の効くものとがある．必ず指定する必要があるのは，名前欄のジョブ名と命令欄の JOB という単語だけである．オペランドで指定するパラメータはすべて任意選択の効くものである．ただし，定位置パラメータである会計情報とプログラマー名は，**ユーザーの会計ルーチン**（user accounting routine）を組み込むことによって，必須のパラメータとすることができる性格も持っている．このほかのパラメータは，すべて省略可能である．省略した場合には，システムがあらかじめ定めている値ないしはオペレーティング・システムの生成時に指定した値が自動的にとられるようになっているものが多い．

以下で，JOBステートメントのオペランドで指定することのできるパラメータのうちの主要なものに関して説明を加えてゆこう．

1） 会計情報の指定

会計情報（accounting information）とは，そのジョブの処理に要した時間や印刷用紙とか穿孔カード枚数などにもとづいて使用料金を算定し，どこに賦課するのかといった処理に必要となる情報である．たとえば，賦課部門コード，プロジェクト番号，請求月日，賦課部門名，賦課部門の場所やプロジェクトの責任者名などが会計情報として与える必要のあるものである．

会計情報は，先述したように使用するシステムで記入すべきかどうかが定め

```
1.  //JOB1   JOB   790
2.  //JOB2   JOB   (790, 02-18-74, 4014B, HARVEY-MAX)
3.  //JOB3   JOB   (790,, 4014B, HARVEY-MAX)
4.  //JOB4   JOB   (790,,, HARVEY-MAX)
```

図 7-45　会計情報の考え方

られている．ここで指定した情報をもとに，ユーザーがオペレーティング・システムにあらかじめ組み込んだ会計ルーチンで処理するのである．したがって，会計ルーチンが組み込まれていないなら，この情報を特に指定する必要はない．会計ルーチンが組み込まれていない場合でも，単にジョブの文書化のねらいのために指定することもあるが，この場合にはシステムでの処理情報としては全く使われない．

会計情報は，JOBステートメントのオペランドで指定する最初の定位置パラメータである．2つ以上のサブパラメータで構成することも許されるが，この場合には，つぎに示すようにかっこまたは引用符で囲む必要がある：

([口座番号] [，他の会計情報，……])

記号〔　〕は，この記入項目が省略できることを示している．口座番号，つまり，実行したジョブの料金を賦課する部門のコードだけはサブパラメータの最初に書くように定められている．そのあとの追加情報に関しては，すべて組込まれた会計ルーチンの処理の仕方に依存する．なお，パラメータが口座番号だけの場合には，かっこや引用符で囲む必要はない．

図 7-45を参照していただきたい．会計情報の与え方の例を4つ示している．1.の例では，口座番号だけしか与えていないので，かっこは不要である．2.の例は，4つのサブパラメータを必要とする会計ルーチンが組み込まれていることを意味している．サブパラメータは，賦課部門コード，請求年月日，プロジェクト番号，プロジェクト責任者名の順番に指定することがあらかじめ約束されていると仮定している．サブパラメータが2個以上あるので，当然かっこで囲む必要がある．3.の例は，2.の場合に同じであるがこのジョブでは請求年月日を省略している．したがって，2番目のパラメータを省略したことを示すため

に，最初のコンマの直後に別のコンマを入れてその旨を知らせている．4.の例では，請求年月日とプロジェクト番号の両方を省略している．そこで，もう1個余計のコンマを挿入してこのことを示している．

なお，ジョブ会計情報の桁数は，サブパラメータを区切るためのカンマも含めて142文字以下と定められている．

2)　プログラマー名の指定

プログラマー名も，使用するシステムで指定すべきかどうかが定められている．プログラマー名は，やはり会計ルーチンで会計情報の一部として利用される．プログラムを書いた責任者を明確にするためのものであり，会計ルーチンで処理しない場合でも，指定しておいた方が好都合であることが多い．プログラマー名の指定の仕方は，つぎの3点に要約される：

- プログラマー名は，JOB ステートメントの2番目の定位置パラメータとして指定する．
- 会計情報が省略されている場合には，プログラマー名の直前にカンマを入れて示す必要がある．
- プログラマー名は20桁以内でないといけない．名前の中にピリオド以外の特殊文字が入るときには，引囲符で名前を囲む必要がある．

プログラマー名の与え方の例を示したのが，**図7-46**である．**1.**の例では，会計情報が省略されているので，2番目の定位置パラメータであることを表示するために最初にカンマを入れる必要がある．他の4つの例では，すべて会計情報が示されているので，カンマの直後にプログラマー名を記入している．

1.	//JOBZERO	JOB	, J. EMURA	
2.	//JOB1	JOB	790, A. SATO	72桁目 ↓
3.	//JOB2 //	JOB	(790, 02-18-74, 4014B, HARVEY-MAX), K.TANAKA	C
4.	//JOB3	JOB	(790,, 4014B, HARVEY-MAX), T. KATO	
5.	//JOB4	JOB	(790,,, HARVEY-MAX), YAMAMOTO	

図7-46　プログラマー名の与え方

JOBステートメントにおける定位置パラメータは，以上の会計情報とプログラマー名の２つだけである．これ以外のパラメータはすべてキーワード・パラメータである．

3) メッセージ・レベルの指定

ジョブ・スケジューラーは，ジョブ出力の一部としてシステム出力装置上にジョブ制御ステートメントと関連メッセージを表示してくれる．つぎの形式のパラメータの組合せで，これらの出力の範囲をジョブ・スケジューラーに要求することが出来る：

MSGLEVEL＝([ステートメント] [，メッセージ])

ステートメント欄では，どの範囲のジョブ制御ステートメントを，ジョブの出力としてシステム出力装置上に書き出すかを示す．一方，メッセージ欄では，どのような条件で割当て／終了メッセージを書き出すかを指示する．割当て／終了メッセージとは，開始／終止プログラムがジョブ・ステップ開始時に入出力装置をどのように割り当てたかのメッセージと，ジョブ・ステップが終了したときのデータ・セットの状況や後始末の様子を示すメッセージのことである．それぞれのパラメータでどういう指定ができ，どういう意味を表わすかを表 7-5 で示しているので参照されたい．なお，最初のサブパラメータを省略した場合には，その旨を示すためのカンマが必要である．つまり，MSGLEVEL＝(，1) のように書く．また，２番目のサブパラメータを省略する場合は，かっこもカンマも書く必要はない．したがって，MSGLEVEL＝2 といった指定でよい．１番目のパラメータが省略されると１が，２番目のパラメータが省略されると０が与えられたと同様に解釈される（表 7-5)．

この MSGLEVEL パラメータの記入例を示したのが，図 7-47である．**1.** は，ジョブ制御ステートメントとすべての割当て／終了メッセージを書き出すよう要求している．**2.** は，ジョブ制御ステートメントのうちの JOB ステートメントだけとすべての場合の割当て／終了メッセージの書き出しを要求している．**3.** では，ステートメント・サブパラメータだけを指定している．したがって，

	指定の意義	パラメータ	パラメータの意味
ステートメント欄	どの範囲のジョブ制御ステートメントをジョブの出力として書き出すかを指定する.	0	JOB ステートメントだけを出力する.
		1	全てのジョブ制御ステートメントとSYS 1. PROCLIB から呼び出されたカタログ式プロシージャーのステートメントの双方を出力する.
		2	全てのジョブ制御ステートメントを出力する.
メッセージ欄	どの範囲の割当て／終止メッセージをジョブの出力として書き出すかを指定する.	0	ジョブが異常終止しないかぎり，割当て／終了メッセージを出力しない．異常終止の場合には，割当て／終了メッセージを出力する.
		1	すべての割当て／終了メッセージを出力する.

表 7-5　MSGLEVEL パラメータの意味

```
1.  //JOBZERO  JOB   , J. EMURA, MSGLEVEL=(2, 1)
2.  //JOB 1    JOB   790, A. SATO, MSGLEVEL=(0, 1)
3.  //JOB 4    JOB   (790, , , HARVEY-MAX), YAMAMOTO,          ×
                     MSGLEVEL=2
4.  //JOB 5    JOB   MSGLEVEL=(, 1)
```

図 7-47　MSGLEVEL パラメータの例

ジョブ制御ステートメントと異常終止したときだけの割当て／終了メッセージを出力するように要求していることになる．最後の例では，逆にステートメント・サブパラメータを省略している．この場合には，**MSGLEVEL=(1, 1)** と指定したのと同様の効果が得られる.

4)　ジョブの終了条件の指定

２つ以上のジョブ・ステップからなるジョブでは，２番目以降のジョブ・ステップが実行できるかどうかは，先行ジョブ・ステップが正常に実行されたかどうかに依存していることはいうまでもない．たとえば，コンパイル／連係編集／実行という一連のジョブ・ステップからなるジョブの“実行”ステップは，先行するコンパイルと連係編集のステップの実行状況いかんによって，実行で

きるかどうかが決定されよう．もちろん，コンパイル時に発見されるエラーのうちには，“実行”ステップに何ら影響を及ぼさない程度のエラーもある．ところが，ジョブ・スケジューラーがとっているアプローチの仕方は，特に何も指定がないと，あるジョブ・ステップが異常終止するとその後のステップの実行を中止する（一般に“flush”という）方法をとっている．この方法だと，先行ジョブ・ステップで何か異常が生じると，後続ステップは必然的に実行されないわけで，都合が悪いときがしばしばある．そこで，このようなジョブの早期終了の条件を修正したい場合が起るのは当然といえよう．このために，処理プログラムで，実行中の異常の度合いを示す**戻りコード**（return code）をレジスター15に入れて，プログラムの実行が終ったときの状態をジョブ・スケジューラーに知らせることを可能にしている．ジョブ・スケジューラーは，処理プログラムから返された戻りコードと JOB ステートメントの中で指定された値とを比較したうえで，ジョブの続行が可能かどうかを判定する．このための値を設定するために COND パラメータを使用する．COND パラメータの記入形式は，つぎの通りである：

COND=([(コード，オペレーター)，… [，(コード，オペレーター)]

コード欄には，0から4095までの整数値を与える．オペレーター欄では，戻りコードと比較する場合の条件を指示する．オペレーターとして使用できる記号は，つぎの6種類である：

GT：戻りコードより大きい
GE：戻りコードより大きいか等しい
EQ：戻りコードと等しい
LT：戻りコードより少さい
LE：戻りコードより少さいか等しい
NE：戻りコードと等しくない

CONDパラメータでは，最低1つの条件から最大8つまでの条件を設定し，テストすることができる．1つの条件だけを指定する場合には，外側のかっこ

```
1.  //JOBZERO   JOB   , J.EMURA, MSGLEVEL=(2, 1),          ×
    //                COND=((50, GE), (60, LT))
2.  //JOB1      JOB   790, A.SATO, MSGLEVEL=(0, 1),        ×
    //                COND=(7, LT)
3.  //TEST      JOB   COND=((20, GE), (30, LT))
```

図 7-48　COND パラメータの例

は不要である.

さて，パラメータがどのように機能するかを例にもとづいて説明しておこう．図 7-48 がそれである.

まず一番目の例からみてゆこう．COND=((50, GE), (60, LT)) と指定してある．これは「50がジョブ・ステップの戻りコードより大きいか等しい」か，または「60がジョブ・ステップの戻りコードより小さい」場合だけ，その後のジョブ・ステップの実行を中止し，バイパスしてしまうことを意味している．つまり，戻りコードが51から60までの範囲のときだけ後続ジョブ・ステップの実行を続けることになる.

2番目の例では，「7が戻りコードより少さい」とき，ジョブを終止してしまうことを示している．したがって，7以下の戻りコードの場合だけ，その後のジョブ・ステップが続行される.

3番目の例は，「20が戻りコードより大きいか等しい」か，または「30が戻りコードより小さい」場合に，後続ジョブ・ステップの実行が中止されることを意味している．戻りコードが21から30までの範囲だとジョブは続行される.

なお，COND パラメータが省略されていると，どれかのジョブ・ステップが異常終止すると，自動的にその後のステップは実行されなくなってしまう.

5)　システム出力装置以外の装置へのメッセージ出力の指定

本章の第2節「ジョブ管理の機能とその構造」の「優先式ジョブ・スケジューリング・システムの機能構造」の項で出力ジョブのクラスについて説明した．出力クラスとしては，36種類までの利用が許されている．出力書出しプログラムは，指定された出力クラスに対してだけ出力サービスをする（どのクラスにサービスすべきかは，START 指令で指示される)．ところで，ジョブ・

```
1.  //JOBZERO   JOB    , J. EMURA, MSGLEVEL=(2, 1),          ×
    //                 MSGCLASS=F
2.  //JOB1      JOB    790, A. SATO, MSGLEVEL=(0, 1),         ×
    //                 COND=(7, LT), MSGCLASS=C
3.  //A1430     JOB    MSGCLASS=L
    //STEP1     EXEC   PRINT
    //OUTPUT    DD     SYSOUT=L
```

図 7-49　MSGCLASS パラメータの例

スケジューラーが出すメッセージとかジョブ制御ステートメントは通常，クラスA，つまり**システム出力装置**（system output device）に出力される．ジョブ制御ステートメントとジョブ・スケジューラー・メッセージをシステム出力装置以外の装置に書き出したい場合には，その旨を JOB ステートメントで指示する必要がある．たとえば，ある種の月報をプリプリント用紙を使ってシステム出力装置で印刷するような時は，その用紙上にジョブ制御ステートメントやメッセージを印刷するのは避けたい筈である．このような場合，ジョブ・スケジューラーの出力と処理プログラムの出力とを別々の装置で印刷するために，JOB ステートメントの MSGCLASS パラメータを使用する．

このパラメータでは，メッセージとジョブ制御ステートメントとを振向ける出力クラスを指定する．クラスの記号としては，AからZの英字と0から9の数字が用いられる．

図 7-49の例をみていただきたい．**1.** ではクラスFに，**2.** ではクラスCにメッセージとジョブ制御ステートメントを振向けることを指示している．一方，**3.** では，メッセージとジョブ制御ステートメント，それに出力データ・セット（DD ステートメントの SYSOUT パラメータを参照のこと）も出力クラスLに振向けることを意味している．

6） ジョブの実行の一時保留の指定

読取／解釈プログラムがジョブを解読したあと直ちにその実行を開始させては困るような性格のジョブが存在することがある．たとえば，データ・セットを作成するジョブとその作成結果を印刷出力するジョブがあると仮定しよう

(ただし，通常の場合だとこのような性格の仕事はジョブ・ステップとして1つのジョブにまとめるが)．この2つのジョブでは，印刷出力のジョブをデータ・セット作成のジョブより先にやるとか，同時併行的にやることはできるとしても，要求する印刷結果は望めない．まず，データ・セットを作成するジョブが行われることが先決である．そのためには，印刷ジョブをデータ・セット作成のジョブが終るまで一時保留にする必要がある．このようなときに用いるのが TYPRUN＝HOLD パラメータである．JOB ステートメントのオペランドで TYPRUN＝HOLD を指定したジョブは，先行すべきジョブの実行が完了したことを確認したとき，保留を解除するための RELEASE 指令をキーインする必要がある．この RELEASE 指令によって，一時保留にされていたジョブは開始／終止プログラムで処理ができるようになる．

7)　領域の大きさの指定

ジョブに必要な主記憶装置は，動的区域の中から開始プログラムが割り当ててくれる．JOB ステートメントで特に指定していないと，あらかじめ定められている大きさの領域を自動的に割り振ってくれる．しかし，定められた大きさよりも大きな領域を必要とするジョブだとか，逆に極く少さい大きさですむようなジョブの場合には，要求する領域の大きさを REGION パラメータで指定する．必要な主記憶域の大きさは，1Kバイト (2^{10}＝1024 バイト）の倍数で指定する．**図 7-50**の例を参照されたい．

8)　ロールアウトの指定

効率という観点から考えた場合，全体の実行時間が比較的少ないにもかかわ

```
1.  //JOBZERO JOB , J. EMURA, REGION=98K
                                                    72桁
2.  //JOB1      JOB  790, ASATO, MSGLEVEL=(0,1),     ↓
                                                      C
                                 COND=(7,LT) REGION=80K
3.  //TEST      JOB REGION=100K
```

図 7-50　REGION パラメータの例

らず異常に大きな記憶域を必要とするようなジョブに，ジョブの開始時から必要な記憶域を与えておくことは得策ではない．そこで，**ロールアウト／ロールイン**（rollout / rollin）の機能が設けられている．この機能は，非常に大きな記憶域を要求するようなジョブに対して，利用可能な記憶域が十分残されていなくてもジョブの開始ができるようにするために設けられた．つまり，ジョブがいったん開始されてから，追加の主記憶域を要求することによって，その記憶域を拡大することができる．要求された記憶域を満すだけの空きがあれば，それが割り当てられるし，もしそれだけの空きがなければ優先順位の低いジョブが占めている記憶域が一時的にあけ渡される．つまり，優先順位の低いジョブの領域の内容がそのまま DASD 上にはき出される（rollout）ことによって，要求を満す方法がとられる．あけ渡した領域が解放されると，DASD 上にはき出されていた内容がそのまま元の記憶域に戻され（rollin），優先順位の低いジョブが再開される．このロールアウト／ロールインは，タスク管理の機能の一部であり，その仕組みの詳細については，第8章「タスク管理の諸機能とその構造」で説明する．

ロールアウトの要求を保障するためには，次の2つのことが必要になる：

- 主記憶域を余分に要求するジョブでは，他のジョブのロールアウトを引きおこすこと指定しておく．
- ロールアウトされても支障のないジョブでは，ロールアウトできる旨を指定しておく．

これらの基準の設定にあたっては，必要な記憶域の大きさ，ジョブの実行時間や実行要件などの事柄をもとに検討する必要がある．ともあれ，JOB ステートメントの中では，つぎのような指定をする：

$$\mathrm{ROLL}=\left(\begin{Bmatrix}\mathrm{YES}\\ \mathrm{NO}\end{Bmatrix}\begin{Bmatrix},\mathrm{YES}\\ ,\mathrm{NO}\end{Bmatrix}\right)$$

最初のサブパラメータでは，該当ジョブがロールアウトできる性格のものであるかどうかを定義する．一方，2番目のパラメータでは，そのジョブがロールアウトを引き起すことができるかどうかを指示する．

```
1.  //JOBZERO JOB , J. EMURA, MSGLEVEL=(2,1), ROLL=(NO, YES)
2.  //TEST    JOB ROLL=(YES, YES), REGION=100 K
```

図 7-51　ROLL パラメータの例

図7-51の例を参照していただきたい．**1.** の例では，このジョブ内のジョブ・ステップをロールアウトすることはできないが，他のジョブのステップを必要ならロールアウトさせることができることを意味している．**2.** の例では，このジョブのジョブ・ステップはロールアウトすることができるし，しかもあるジョブ・ステップで 100K を越える記憶域を必要とする場合には，他の優先順位の低いジョブのステップのロールアウトを引き起すことができることを指示している．

9）ジョブの実行時間の制限指定

ジョブがあらかじめ想定した CPU の使用時間を越えたり，30分以上連続して待ち状態に入ったりしたとき，そのジョブの実行を打ち切るための指定ができる．これに使うのが TIME=(分，秒) のパラメータである．たとえば，あるジョブがCPU時間を 9 分 4 秒以上は使用しないと想定されるなら，TIME=(9, 4) と指定すればよい．この指定によって，9 分 4 秒を越えて CPU 時間を使用するとそのジョブは打ち切られる．また，実行中に30分以上連続して待ち状態が続くとやはりジョブが中止される．TIME=1440 と指定すれば，制限時間はないものとみなされるし，30分以上の待ち時間があってもジョブは打ち切られなくなる．TIME パラメータを省略すると，そのシステムで定められている制限時間が適用される．

図 7-52 に TIME パラメータの指定例を示している．**1.** はジョブが使用する CPU の最高時間を 12分10秒としている．**2.** では，30 秒だけである．分の部分を省略するとカンマが必要である．**3.** の場合は，制限時間は 5 分である．秒の指定がないときはかっこは不要である．**4.** の例は，先述したように無制限である．したがって，30分間の連続待ち状態も無視される．

```
1.  //JOBZERO   JOB , J. EMURA, MSGLEVEL=(2, 1), TIME=(12,10)
2.  //JOB1      JOB 790, A. SATO, COND=(7, LT), TIME=(, 30)
3.  //JOB4      JOB (790,,, HARVEE-MAX),MSGLEVEL=2,TIME=5
4.  //SAMPLE    JOB TIME=1440
```

図 7-52　TIME パラメータの例

10)　優先順位とジョブ・クラスの指定

ジョブの優先順位も，JOB ステートメントのオペランドで指定する．PRTY パラメータで 0 から13の数字を使って優先順位を指定する．0 が，最も優先順位が低いことを示す．逆に13は，最も優先順位の高いジョブである．なお，このパラメータを省略した場合には，使用するシステムで定められている標準の優先順位がそのジョブに適用される．

ジョブ・スケジューラーがジョブの開始順序を決定するのに使うもう 1 つの要素が**ジョブ・クラス**（job class）である．ジョブ・クラスごとにジョブを入力待行列に入れ，ジョブ・クラスに対応して開始プログラムが機能することは，本章の第二節「ジョブ管理の機能とその基本構造」の「優先式ジョブ・スケジューリング・システムの機能構造」の項ですでに説明した通りである．入力ジョブ待行列のどのクラスにジョブを割り当てるかは，CLASS パラメータで指定する．ジョブ・クラスは A から O までの16種類が指定可能である．

優先順位とジョブ・クラスの指定例を**図 7-53**に示している．**1.** の例では，ジョブ・クラス C で，ジョブ選択の優先順位を 8 としている．**2.** の場合には，ジョブ・クラスが A で，優先順位が 4 のジョブである（CLASS パラメータが省略されると，自動的にクラス A とみなされる）．**3.** の場合には，ジョブ・クラ

```
1.  //JOBZERO JOB , J. EMURA, PRTY=8, CLASS=C
2.  //JOB1    JOB 790, A. SATO, TIME=(, 30), PRTY=4
3.  //JOB4    JOB (790,,, HARVEY-MAX),MSGLEVEL=2, CLASS=C
4.  //JAN     JOB CLASS=M, PRTY=10
```

図 7-53　PRTY と CLASS パラメータ

//名前欄	命令欄	オ ペ ラ ン ド	P/K	備　考
//ジョブ名	JOB	(〔口座番号〕〔, 追加の会計情報, •••〕)	P	必須オペランドにすることも可能
		〔プログラマー名〕	P	必須オペランドにすることも可能
		〔CLASS=ジョブクラス〕	K	A〜O のクラス名を使用
		〔COND=((コード, オペレータ), ••••)〕	K	最大8個までテスト可能
		〔MSGCLASS=出力クラス〕	K	A〜Z, 0〜9 のクラス名を使用
		〔MSGLEVEL=({0/1/2}〔, 0/, 1〕)〕	K	
		〔PRTY=優先順位〕	K	0〜13 の優先順位番号を使用
		〔REGION=サイズ〕	K	K単位で指定
		〔ROLL=({YES/NO}{, YES/, NO})〕	K	ロールアウト／ロールインの指定
		〔TIME={(分, 秒)/1440}〕	K	ジョブの CPU 使用時間の制限
		〔TYPRUN=HOLD〕	K	ジョブ待行例にあるジョブの保留

凡例：P—定位置パラメータ
K—キーワード・パラメータ
{ }—いづれか1つを選択
〔 〕—任意選択可能，2行以上が囲まれている場合にはどれか1つの選択指定

表 7-6　JOB ステートメントの要約（主要なパラメーターだけ）

スCが割り当てられている．優先順位は，そのシステムの標準のものが適用される．4. の例では，ジョブ・クラスMで優先順位10が割り当てられている．

以上が，JOBステートメントで指定できる主要なパラメータである．これまで説明してきたパラメータの要約を表 7-6 に掲げておいたので参考にしていただきたい．

3.　EXECステートメントでの主要な指定内容

ジョブ・ステップの実行を開始するためには，開始プログラムに実行するプログラム名とそのステップで必要とする入出力の要件を知らせる必要がある．

実行するプログラム名は EXEC ステートメントで，入出力の要件は DD ステートメントで知らせる．このことは，すでに説明した通りである．この種の情報を開始プログラムに伝達する１つの方法は，入力ジョブの流れの中に直接 EXEC ないしは DD ステートメントを入れて置くやり方である．別の方法は，これらステートメントの集まりをカタログ式プロシージャーとして，SYS1. PROCLIB に収容しておき，そのプロシージャーの名前を指定した EXEC ステートメントで入力ジョブの流れの中に呼び込むやり方である．読取／解釈プログラムは，EXEC ステートメントの中にプロシージャー名が指定されていると，そのプロシージャーを構成する EXEC ステートメントと DD ステートメントを PROCLIB から取り出し，開始プログラムで利用できるようにしてくれるのである．

1） プログラム名ないしはプロシージャー名の指定

前述したように，EXEC ステートメントの主要なねらいはジョブ・スケジューラーにそのジョブ・ステップで使用するプログラム名ないしはプロシージャー名を知らせることにある．プログラム名は，キーワード PGM＝を使用して指定する．一方，プロシージャー名は，キーワード PROC＝を使用して指定してもよいし，定位置パラメータの形式で指定することもできる．PGM パラメータにしろ，PROC パラメータにしろキーワード・パラメータの形式をとっているが，実際には定位置パラメータであって，必ずオペランドの最初に記入する約束になっている．

図 7-54で指定例を示している．**1.** ではプログラム名を，**2.** と**3.** ではプロシージャー名を指定している．

図 7-54では，いずれの EXEC ステートメントも名前欄にジョブ・ステップ

```
1. //          EXEC    PGM=IEHPROGM
2. //          EXEC    PROC=SCRATCH
3. //          EXEC    SCRATCH
```

図 7-54 ステップ名のない EXEC ステートメントの例

```
1.  // STEPONE  EXEC   PGM=IEHPROGM
2.  // A 123456 EXEC   PROC=SCRATCH
3.  // LAST     EXEC   SCRATCH
```

図 7-55　ステップ名のある EXEC ステートメントの例

```
1. // CREATE    EXEC   PGM=IEWL                                      72桁目
   // SYSLMOD   DD     DSNAME=&&PARTDS (PROG),                          ↓
                                            UNIT=2311,              ×
   //                DISP=(MOD, PASS), SPACE=(1024, (50, 20, 1))
   // EXECUTE   EXEC   PGM=*. CREATE. SYSLMOD
2. // STEP 2    EXEC   PGM=UPDT
   // DDA       DD   DSNAME=SYS 1.LINKLIB(P40),DISP=OLD

   // STEP 3    EXEC   PGM=*. STEP 2. DDA
```

図 7-56　プログラム名としてデータ定義名を使用する例

名を記入していない．このようにステップ名を省略することができる．しかし，ステップ名は与えておいた方がよい．というのは，ステップ名を与えておくとメッセージ表示の利点があるからだ．マスター・スケジューラーが書出す多くのメッセージではステップ名を含めてくれる．ステップ名が与えられていないと，その部分がブランクのメッセージになってしまうので何かと不便である．したがって**図 7-55**に示したようにステップ名を与えておいた方がよい．

ステップ名を与えることのもっと重要な理由もある．あるジョブ・ステップを後続のジョブ・ステップから参照する必要が生じるケースである．この場合の具体例を1つ示しておこう．別の具体例は，それぞれのパラメータの指定個所ででてくるのでここでは割愛しておく．**図 7-56**を参照していただきたい．この例は，PGM パラメータでの特別な指定方法である．プログラム名の代りに先行するジョブ・ステップで定義されたデータ定義名を用いる方法である．*．ステップ名．データ定義名の形式で指定する．ステップ名で DD ステートメントが定義されているステップの名前を示す．データ定義名では，実行するプログラムが貯えられている区分データ・セットのデータ定義名を指示する．図7-56の**1.** では，CREATE ステップで連係編集プログラム（IEWL）が作り出した

ロード・モジュールを EXECUTE ステップで実行する場合を示している．連係編集ステップで作られたロード・モジュールのプログラムは，データ定義名 SYSLMOD で定義されている区分データ・セットに収容されるので，そのデータ定義名を使って使用するプログラムを指示している．一方，**2.** の例では，STEP 2 でプログラム名 P 40が SYS 1. LINKLIB に貯えられるので，それを使って STEP 3 の実行をすることを指示している．

EXEC ステートメントにおいては，プログラム名またはプロシージャー名が必須のパラメータであり，その他のパラメータは任意選択ができる．任意選択が効く主要なパラメータは，つぎの 5 種類である：

- 先行ステップの実行状態にもとづいて，特定ジョブ・ステップを実行するかどうかを指定（COND パラメータ）
- そのジョブ・ステップで実行するプログラムで必要とする特別な情報の指定（PARM パラメータ）
- ジョブ・ステップ用の会計情報（ACCT パラメータ）
- ジョブ・ステップの実行時間の制限指定（TIME パラメータ）
- ジョブ・ステップで必要とする領域の大きさの指定（REGION パラメータ）

これらパラメータは，すべてキーワード・パラメータであり， PARM パラメータ以外は JOB ステートメントで指定することのできたパラメータと同じような機能をもっている．以下でそれについて簡単に説明しよう．

2）バイパス条件の指定

沢山のジョブ・ステップから構成されるようなデータ処理業務では，条件次第でそのうちのいくつかのステップは実行しなくてよいといった事態が生じることがある．たとえば，図7-57の受取勘定業務のようなジョブでは，ステップ 1 で支払済み勘定レコードがみつからなければ，ステップ 2 を開始しても無駄といえる．逆に，滞納勘定レコードがないのにステップ 3 を実行するのも意味がない．特別な場合には，こういった事態が発生することが当然予想される．

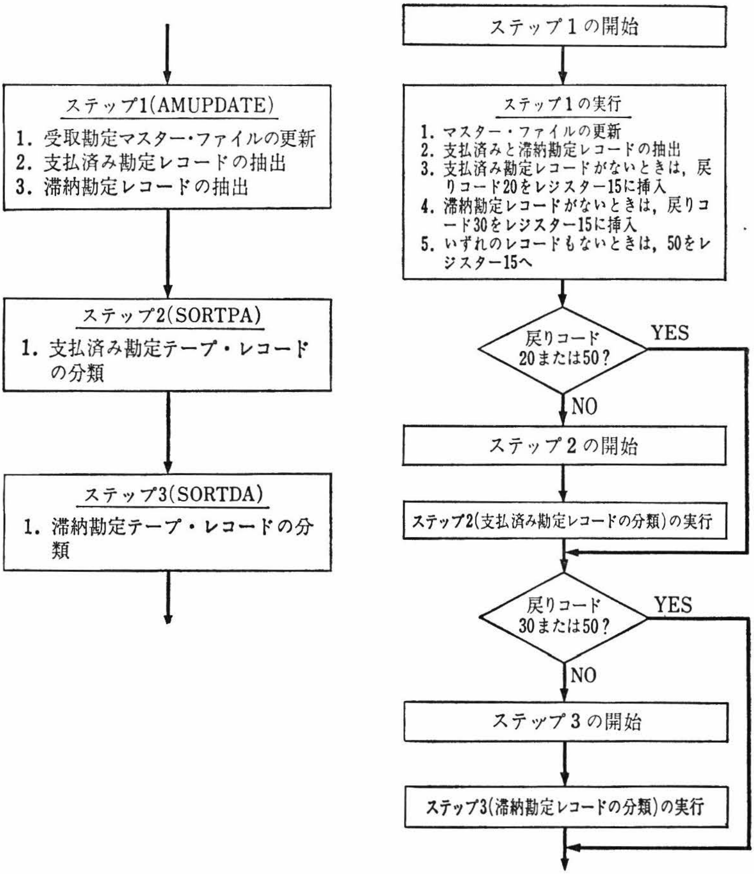


図 7-57　特別な条件でジョブ・ステップをバイパスする必要のある例

このような場合，先行ステップで発見された条件にもとづいて，ジョブ・ステップをバイパスすべきかどうかの判断をする必要がある．これを行うためには，**図 7-57**の左側で示しているように2つのことが要求される．まず，処理プログラムでは，そのプログラムの実行中に見つけ出した状態を**戻りコード**(return code) として制御プログラムに知らせる．一方，EXEC ステートメントでは，そのステップをバイパスするかどうかのテスト条件を与える必要がある．

このやり方は，JOB ステートメントで戻りコードの結果ジョブ全体を終了

```
        ≀
//STEP 1  EXEC  PGM=AMUPDATE
        ≀
        ≀
//STEP 2  EXEC  PGM=SORTPA, COND=((20, EQ, STEP 1),
                                  (50, EQ, STEP 1))
        ≀
        ≀
//STEP 3  EXEC  PGM=SORTDA, COND=((30, EQ, STEP 1),
                                  (50, EQ, STEP 1))
        ≀
```

図 7-58 条件コードで戻りコードをテストする例

させる要求を出す場合と同じである．つまり，COND=(　)のキーワード・パラメータを使用する．異るのは，バイパス条件が生じるジョブ・ステップを示すために，3番目のサブパラメータとしてステップ名を与える必要があることだ．たとえば，STEP 1 で戻りコードとして19以下がレジスター15に入れられていると STEP 2 をバイパスする場合だと，COND=(20, GE, STEP 1) と指定する．つまり，「20の方が STEP 1 で戻されたコードより大きいか等しい場合には STEP 2 をバイパスしろ」ということになる．もし，バイパスを決定するのに先行するすべてのジョブ・ステップをテストする必要がある場合には，ステップ名を省けばよい．たとえば，先行するどれかのジョブ・ステップで20，30ないしは40の戻りコードが示されると，当該ジョブ・ステップをバイパスして欲しいとすると，COND=((20, EQ), (30, EQ), (40, EQ)) と指定すればよい．ステップ名が与えられていないと，すべての先行ステップの戻りコードがテストされることを意味しているのである．

図 7-58に図7-57の条件にもとづいた COND パラメータの指定例を示しているので参照されたい．

なお，先行ジョブ・ステップが異常終止したときだけ (ONLY) そのジョブ・ステップを実行したいとか，先行ステップが異常終止しても (EVEN) そのジョブ・ステップは実行したいといった場合がある．前者の場合には，COND=ONLY と指定し，後者の場合には，COND=EVEN と指定すればよい．これらは，戻りコードのテストと併用することもできる．

3) 会計情報の指定

ジョブの口座番号単位でコンピュータ使用料金を請求するための情報として会計情報を与えることができた．これと同様にジョブ・ステップ単位で使用料を請求することも可能だ．このために，EXECステートメントでは，ACCTパラメータを使用する．このACCTパラメータを処理するのはあくまでユーザーが組込んだ会計ルーチンである．したがって，このパラメータの中で指定する内容だとか指定順序は会計ルーチンの処理方法によってきまる．

ともあれ，会計情報の指定の仕方は，つぎの5点に要約できる：

- キイワード・パラメータ ACCT を使って指定する．
- 指定できる最大桁数は，142桁である．
- 会計情報のサブパラメータが1個だけの場合は，かっこで囲む必要はない．
- 特殊文字が含まれている場合は，そのサブパラメータは引用符で囲む．
- サブパラメータ自体は定位置パラメータとみなされる．したがって，サブパラメータのうちのいくつかを省略する場合には，その旨をカンマで表示する必要がある．

会計情報の指定例は，図7-59の中に示してあるので参照されたい．

4) 処理プログラムに受渡すパラメータの指定

EXECステートメントで，ジョブ・ステップで実行されるプログラムに情報を渡すことができる．これを指定するのが，PARMキーワード・パラメータである．PARMパラメータの記入形式は次の通りである：

PARM＝値

この値が，ジョブ・ステップで実行するプログラムに渡す最高100桁の情報である．PARMの値が2つ以上のサブパラメータから成る場合には，その値をかっこまたは引用符で囲む必要がある（**図7-59**の例を参照のこと）．引用符ないしはかっこは処理プログラムには渡されないが，サブパラメータを区切っ

```
                                                               72桁目
                                                                 ↓
1.  //STEP 3    EXEC  PGM=PGMA 25, COND=(25, LE, STEP 1),        C
    //                ACCT=(2978787, 03-01-74), PARM=(3, 28)
2.  //STEP 12   EXEC  PGM=PROG 3, COND=((20, EQ, STEP 5),
                                       (30, EQ, STEP 5)),        C
    //                ACCT=(09240, , BIGPROJ, EMURA),
                                       PARM=(1, 0, ABC)
```

図 7-59　ACCT と PARM パラメータの与え方

ているカンマはそのまま渡される．

処理プログラムが PARM パラメータの情報を受け取るのは，ちょうど制御プログラムから処理プログラムにパラメータが受渡されるのと同じ要領である．つまり，EXEC ステートメントでパラメータを指定すると，開始プログラムはそのパラメータを所定の主記憶域に貯え，その記憶域のアドレスを汎用レジスター1に入れて処理プログラムに知らせる．処理プログラムでは，汎用レジスター1に入っているアドレスを用いてパラメータを取り出し処理に使うことができる．

5)　ジョブ・ステップの実行時間の制限指定

プログラム・テストのさいに生じるやっかいな問題に，プログラムが**ループし放しの状態** (endless loop) に入ることがある．コンパイルと連係編集のステップで誤りが生じなくても，実行段階に論理的ミスのため終りなきループに入ることは予想される．このようなコンピューター時間を無駄にしてしまう状態を回避するために，ジョブ・ステップのコンピュータ利用時間を制限することができる．EXEC ステートメントの中の TIME キーワード・パラメータで制限時間を指定すればよい．指定の仕方は，JOBステートメントでのそれと全く同じ要領である（図 7-60 参照）．

```
1.  //STEP 1    EXEC   PGM=REDONE, TIME=(12, 10)
2.  //FOUR      EXEC   PGM=REDFOUR, TIME=(, 30)
3.  //TEST      EXEC   PGM=CALCPROG, TIME=5
```

図 7-60　TIME パラメータの与え方

指定された時間内にそのジョブ・ステップが終了しないときは，そのジョブ全体を異常終止（ABEND）させてしまう．

6） ジョブ・ステップごとの領域の大きさの指定

JOBステートメントで領域の大きさが指定できるが，この場合にはそのジョブを構成するそれぞれのジョブ・ステップに同じ大きさの領域が適用される．ごくわずかな記憶域しか使用しないステップに対しても指定された領域が割り当てられるので無駄が生じてしまう．したがって，各ステップで必要とする記憶域の大きさにかなりの差があるような場合，全ステップに同じ大きさの領域を確保するのは得策ではない．たとえば，コンパイル／連係編集／実行といったステップからなるジョブを考えてみよう．コンパイルには200K，連係編集には44K，そして実行には36Kバイトの記憶域が見積られたとする．JOBステートメントで領域を要求するとしたら200Kバイトになる．しかも，実行ステップにはわずか36Kバイトしか利用しない．未使用の部分が別のジョブの実行に利用できたら大助かりである．

ということから，EXECステートメントのREGIONパラメータを使ってステップごとに領域の大きさを指定することができるようになっている．さきの例だと，それぞれのステップのEXECステートメントでREGION＝200K，REGION＝44K，REGION＝36Kと指定すればよい（図7-61参照）．

7） ロールアウトの指定

JOBステートメントでROLLキーワード・パラメータを指定すると，そのジョブの中の全ステップに影響を及ぼしてしまう．そこで，各ジョブ・ステッ

```
             ≀
//COMPILE  EXEC  PGM=IEUASM, REGION=200K
             ≀
             ≀
//LINKED   EXEC  PGM=IFWL, REGION=44K
             ≀
             ≀
//GO       EXEC  PGM=*. LINKED. SYSLMOD, REGION=36K
             ≀
```

図7-61　REGIONパラメータの与え方

```
1. //RAND  EXEC  PGM=LOOKUP, REGION=50K, ROLL=(YES, YES)
2. //FILL  EXEC  PGM=PLUS, ROLL=(YES, NO), REGION=100 K
3. //UP    EXEC  PGM=P 2165, ROLL=(NO, YES)
```

図 7-62　ROLL パラメータの与え方

プごとに異ったロールアウト条件の指定をしたい場合には，JOBステートメントではなく，EXEC ステートメントの ROLL パラメータを用いる．EXEC ステートメントの ROLL パラメータを用いると，そのステップだけにしか効力が及ばないからである（図 7-62参照）．

以上が EXEC ステートメントで指定できる主要なパラメータである．ここで説明したパラメータを表 7-7 にまとめているので参考にして欲しい．なお，プロシージャー・ステップ名を用いる方法については後述する．

4. DD ステートメントでの主要な指定内容

処理プログラムの実行を開始するまえに，そのプログラムで必要とするデータ・セット，ボリュームおよび直接アクセス・ボリュームのスペースなどを開始プログラムが割り当てる必要がある．開始／終止プログラムにこの種の情報を提供するのが，DD ステートメントの役割りである．下記にまとめたような情報が，開始／終止プログラムおよびアクセス方式ルーチンが必要とするものである：

1. 一時的データ・セットでない場合のデータ・セット名（一時的データ・セットの場合は，システムが自動的に名前を割り振る）
2. DASD 上に作成するデータ・セットのスペースの大きさ
3. データ・セットの状況および使用後の後始末の仕方
4. 装置のタイプ（磁気テープ装置，磁気ディスク装置など）
5. 特定のボリュームを要求する場合のボリューム連続番号
6. 磁気テープ・データ・セットの場合のラベル情報
7. 入出力バッファの要件
8. データ・セットを複数のプログラムで同時に共用できるかどうか

//名前欄	命令欄	オペランド	P/K	備考
//〔ステップ名〕	EXEC	{PGM={プログラム名 *. ステップ名. データ定義名 *. ステップ名. プロシージャー・ステップ名. データ定義名} 〔PROC=〕プロシージャー名}	P	プログラムまたはプロシージャー名を指定.
		〔ACCT=(会計情報, ‥‥) ACCT. プロシージャー・ステップ名=(会計情報, ‥‥)〕	K	ジョブ・ステップへの会計情報.
		〔COND=({(コード, オペレーター) (コード, オペレーター, ステップ名) (コード, オペレーター, ステップ名. プロシージャー・ステップ名)}, …〔, 〕[EVEN / ONLY]) COND. プロシージャー・ステップ名=({(コード, オペレーター) (コード, オペレーター, ステップ名) (コード, オペレーター, ステップ名. プロシージャー・ステップ名)}, …〔, 〕[EVEN / ONLY])〕	K	最大8個のテストが可能. EVEN または ONLY を指定したときは7個までのテストが可能.
		〔PARM=値 PARM. プロシージャー・ステップ名=値〕	K	かっこないしは引用符で値を囲むこと.
		〔REGION=サイズ REGION. プロシージャー・ステップ名=サイズ〕	K	K単位で指定.
		〔ROLL=({YES / NO}〔, YES / , NO〕) ROLL. プロシージャー・ステップ名=({YES / NO}〔, YES / , NO〕)〕	K	ロールアウト／ロールインの指定.
		〔TIME={(分, 秒) / 1440} TIME. プロシージャー・ステップ名={(分, 秒) / 1440}〕	K	ジョブ・ステップの CPU の使用制限時間を指定.

凡例： P—定位置パラメータ
K—キーワード・パラメータ
{ }—いずれか1つを選択
〔 〕—任意選択可能, 2行以上が囲まれている場合はどれか1つの選択指定.

表 7-7 EXEC ステートメントの要約（主要なパラメータだけ）

9. ジョブ・ステップで使用するデータ・セットに対して同じ入出力チャネルを割り当てるかどうか

以上のような情報をDDステートメントのオペランドで必要に応じて指示することができる．それぞれの指定内容と指定の仕方を以下で説明する．

1) DISP パラメータの指定

DISP で指定するパラメータでは，開始プログラムに使用データ・セットの性質を知らせる，ジョブ・ステップが正常に終了した場合のデータ・セットの後始末の仕方を終止プログラムに知らせる，および異常終止した場合のデータ・セットの後始末の仕方を知らせるという3つのねらいのために用いられる．このパラメータの一般形式を**図 7-63**に示している．

① データ・セットの状況指定

DISP の最初に指定するサブパラメータは，データ・セットの性格ないしは状況を開始プログラムに知らせる．このサブパラメータでは，つぎの4つのものから選択して指定する必要がある：

1. NEW：このジョブ・ステップを開始する前には，当該データ・セットは存在しないことを意味する．つまり，このステップで新規に作成される性質のデータ・セットであることを知らせる．
2. OLD：このジョブ・ステップを開始する前から存在するデータ・セットであることを意味する．換言すれば，入力される性質のデータ・セ

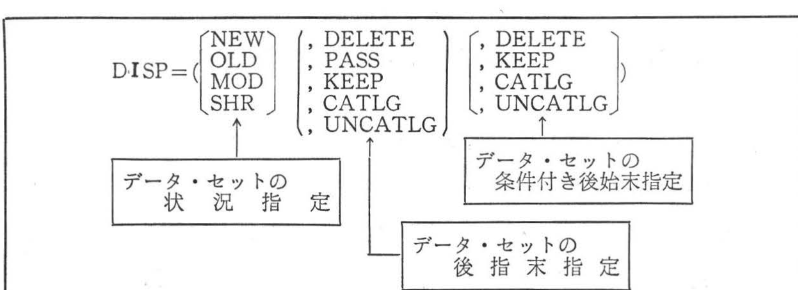


図 7-63 DISP パラメータの一般形式

ットであることを示す.

3. MOD：レコードが追加される性質のデータ・セットであることを意味する．ジョブ・ステップが開始される前から存在しているデータ・セットであるという点では OLD に似ている．しかし，このデータ・セットが存在しないときは，開始プログラムは NEW とみなす点が大きく異っている.
4. SHR：データ・セットは OLD と同じである．ただし，このデータ・セットは，複数個のプログラムで同時に共用できることを意味している．この指定をしていない，つまり OLD と指定したデータ・セットは同時に共用することのできないデータ・セットとみなされてしまう.

② データ・セットの後始末指定

DISP の2番目のサブパラメータは，ジョブ・ステップが正常に終了したときに当該データ・セットの後始末をどうすべきかを終止プログラムに教えるためのものである．このサブパラメータでは，つぎの5つのうちのいずれかが指定できる：

1. KEEP：終止プログラムがデータ・セットの割当てを解除したあとも当該データ・セットの DSCB はそのまま VTOC に保持されることを意味する．つまり，データ・セットは保存される性質のものである.
2. PASS：終止プログラムは，ステップの終りにそのデータ・セットの割当てを解除しないことを意味する．データ・セットが収容されているボリュームは，ジョブ・ステップ間でもマウントされたままである．つまり，このデータ・セットは後続のジョブ・ステップに渡され，利用される性質のものである.
3. DELETE：当該データ・セットの DSCB は VTOC から削除され，データ・セットが占めていた DASD 上のスペースは他に解放されることを意味する．データ・セットを見つけ出すのに，カタログが使われたのならデータ・セット名もカタログから取り除かれてしまう．一方，カタログを使用しないで DD ステートメントの中にボリューム番号と装置のタ

イプを指示して，それでデータ・セットのありかをつきとめた場合だと，DSCB が VTOC から削除されるだけでデータ・セット名がカタログから取り除かれることはない．ともあれ，この指定をするとデータ・セットはステップの終りに削除されてしまう．

4. CATLG：終止プログラムは記入項目（entry）をカタログに登録することを意味する．記入項目には，DD ステートメントで指定したデータ・セットのボリューム連続番号と装置タイプが含まれる．いずれにしろ，データ・セットをカタログしたいときにこの指定をする．なお，カタログの詳細に関しては，第5章の第4節「カタログ・システムの機能と構造」を参照されたい．

5. UNCATLG：データ・セットを参照するためのボリューム連続番号と装置タイプをカタログから取り除いてしまうことを意味する．しかし，データ・セットそのものは削除されない．単にデータ・セットをカタログからはずすだけである．

③ データ・セットの条件付き後始末指定

ジョブ・ステップが正常に終ったときに適用される後始末は，ステップが異常終止したときにもそのまま適用される．しかし，DISP の3番目のサブパラメータを指定すると，これが異常終止のときだけ適用される．このサブパラメータが指定されているときは，2番目のサブパラメータはステップが正常に終ったときにだけ適用されるのである．3番目のサブパラメータの指定項目は，PASS を除いて2番目のサブパラメータのそれと同じである．

④ 省略時解釈

3番目のサブパラメータを省略したときは，2番目のサブパラメータで指定した内容が，ジョブ・ステップの正常終止にも異常終止にも適用されることはさきに説明した通りである．この他にもいろんな省略が許されている．たとえば，DISP パラメータ全体を省略することもできる．この場合には，開始プログラムは NEW とみなし，終止プログラムは DELETE とみなす．つまり，DISP＝(NEW, DELETE) ないしは DISP＝(NEW, DELETE, DELETE) と指

```
1. //DD 1      DD     DSNAME=D 99. GROUP. SIX, UNIT=2311,
                                    VOLUME=SER=111111,      ×
   //                 DISP=(NEW, CATLG, DELETE), SPACE,
                                          =(TRK, (5, 1))
2. //DD 2      DD     DSNAME=FIX, UNIT=2400-1, VOLUME
                                          =SER=09899,       ×
   //                 DISP=(OLD,, DELETE)
3. //STEP 1    EXEC   PGM=FILL
   //DD 1      DD     DSNAME=SWITCH. LEVEL 18. GROUP 12,
                                          UNIT=2311,        ×
   //                 VOLUME=SER=LOCAT 3, SPACE=(TRK,
                                    (80, 15)), DISP=(, PASS)
   //STEP 2    DD     PGM=CHAR
   //DD 2      DD     DSNAME=XTRA, DISP=OLD
   //DD 3      DD     DSNAME=*. STEP 1. DD 1, DISP=(OLD'
                                          PASS, DELETE)
   //STEP 3    EXEC   PGM=TERM
   //DD 4      DD     DSNAME=*. STEP 2. DD 3, DISP=(OLD,
                                        CATLG, DELETE)
```

図 7-64　DISP パラメータの使用例

定したと同じことになる．つぎの3つの例も，サブパラメータの一部を省略した例である：

1. DISP=OLD
2. DISP=(, KEEP)
3. DISP=(, PASS)

最初の例では，開始プログラムに対する情報だけしか指定されていない．この場合には，終止プログラムは最初のサブパラメータが OLD であることから，KEEP を省略時の解釈とする．一方，2番目と3番目の例では，最初のサブパラメータが省略されている．最初のサブパラメータがないときは，開始プログラムは NEW と解釈する．

さて，最後に DISP パラメータの具体的な使用例をみておこう．図 7-64を参照していただきたい．

1. の例では，新規データ・セットの作成であることを開始プログラムに指示し，このステップが正常に終了した場合には，システム・カタログにインデッ

クス記入項目を作りあげることを要求している．このステップが異常な状態で終了した場合だと，データ・セットをカタログする代りに削除すべきことを終止プログラムに要求している．なお，このステップが正常に終了するとシステムが自動的に DD 9 と GROUP に対応するインデックスも作りあげてくれる．

2. の DD ステートメントでは，すでに存在するデータ・セットであることを定義し，このステップが正常に終了したときはデータ・セットを保存するよう要求している（すでに存在するデータ・セットであることから，2番目のサブパラメータが省略された場合にはデータ・セットを保存するものと解釈する）．このステップが異常終止したときはデータ・セットを削除するように3番目のパラメータで要求している．

最後に**3.** の例を説明しておこう．STEP 1 の DD 1 と名付けられた DD ステートメントでは，新規データ・セットであることを指示し，かつこのデータ・セットを後続のジョブ・ステップにパスすることを要求している．STEP 1 が異常終止した場合には，新規データ・セットでかつ条件つき後始末のサブパラメータが省略されているので，このデータ・セットは削除される．STEP 2 の DD 3 と名付けられた DD ステートメントでは，STEP 1 でパスされたデータ・セットを受け取って使用し，さらに後続ステップにパスすることを要求している．STEP 2 が異常終止したときは，条件つき後始末の指定が DELETE であるので削除される．STEP 3 の DD 4 と名付けられた DD ステートメントでは，STEP 1 からパスされてきたデータ・セットを受取って利用し，そのステップの終りでカタログすることを要求している．STEP 3 が異常終止としたときも，条件付き後始末指定を DELETE としているので，データ・セットは削除される．

2) データ・セット名 (DSNAME) の指定

データ・セットには，あとで他のジョブで使用する**非一時的データ・セット** (non-temporary data set) と，あるジョブで作成されそのジョブの中で使用されて削除されてしまう運命にある**一時的データ・セット** (temporary data set) とがある．

非一時的データ・セットは，保存あるいはカタログされて，他のジョブまたは他のジョブ・ステップで使用することができる．一時的データ・セットは，そのジョブ内の後続ジョブ・ステップに渡して使用することができるが，そのジョブ内でのみ有効なデータ・セットである．

したがって，データ・セットを作成するとき，そのデータ・セットを他のジョブで使用するためにカタログあるいは保存しておかなければならないときは，非一時的データ・セットとして定義する必要がある．一方，作成するデータ・セットが，そのジョブ内だけで使用されて削除されるのであれば，一時的なデータ・セットとして定義する．これらのデータ・セットは，次のような形式の DSNAME パラメータで定義する：

- 非一時的データ・セットの場合：DSNAME＝データ・セット名
- 一時的データ・セットの場合：　DSNAME＝&& データ・セット名

① 非一時的データ・セットの指定

非一時的データ・セット名は，修飾された形と修飾されない形がある．つぎにその例を示そう：

- 修飾されたデータ・セット名：ZAIKO. SHUKO. BOLTA
- 修飾されないデータ・セット名：NEWZAIKO

修飾されないデータ・セット名は，1～8桁の文字で構成される．最初の文字だけは，英字である．残りの文字は，英数字であってかまわない．修飾されたデータ・セット名は，磁気テープ・データ・セットの場合には17桁以内，DASD データ・セットの場合だと44桁以内である，これは，第5章「データ管理の諸機能とその構造」の第四章「ボリューム・ラベルとデータ・セット・ラベル」で説明したように，データ・セット・ラベルのデータ・セット名を記録するフィールドが，それぞれ17桁と44桁であることに起因している．ともあれ，修飾されたデータ・セット名は，8桁以内の文字群ごとに，ピリオドを挿入する必要がある．

② 一時的データ・セットの指定

一時的データ・セットのDSNAMEパラメータの指定では，データ・セット名の前に&&を付けて，その旨を知らせる. DSNAME=&&TEMPDATA と指定してデータ・セットを作成すると，後続のジョブ・ステップで，これと同じ名前を使ってデータ・セットを使用することができる.

しかし，一時的データ・セットには，必ずしもDSNAMEパラメータを与えなくてもよい. システムが一時的データ・セットに対して名前を生成してくれるからだ.

DSNAME パラメータを指定しない一時的データ・セットを後続のジョブ・ステップで使用するときは，**図7-65**のような指定をする. STEP 1 でデータ定義名 OUTFL をもつ DD ステートメントで一時的データ・セットの新規作成を指示している. しかし，このDDステートメントではDSNAMEパラメータは指定していない. このデータ・セットを同一ジョブ内の後続ジョブ・ステップ，つまり STEP 2 で使用するときには，DD ステートメントのDSNAMEパラメータで，つぎのように指定している：

DSNAME=*. STEP 1. OUTFL

（STEP 1 ↑ ステップ名，OUTFL ↑ データ定義名）

＊は**逆方向参照**（refer-back）の意味がもたされている. ステップ名では使用するデータ・セットが作成された（ないしは定義された）ステップを表わし，データ定義名では使用するデータ・セットが定義されているDDステートメン

```
//STEP 1    EXEC   PGM=PROG 1
//OUTFL     DD     UNIT=2400, DISP=(NEW, PASS)
            ~
            ~
//STEP 2    EXEC   PGM=PROG 2
//INFL      DD     UNIT=2400, DSNAME=*. STEP 1. OUTEL,
            ~                                  DISP=OLD
```

図 7-65 DSNAME パラメータの指定のないデータ・セットを後続ステップで使用する場合の例

トを指示する.

③ データ・セットの一部を参照するための指定

区分データ・セットのメンバーや**世代別データ群** (generation data group) の特定世代を参照する場合には，データ・セット名をメンバー名や世代番号で修飾する必要がある．例えば，これから作成しようとしている世代を指定する場合には，つぎのようになる：

DSNAME＝JENRSHN (＋1)

一方，3世代前に作成された世代を参照するためには，つぎのようになる：

DSNAME＝JENRSHN (－3)

世代別データ群の構造とその指定に関する具体例は，第5章の第4節「カタログ・システムの機能と構造」で詳述しているので参考にして欲しい.

区分データ・セットの特定メンバーを参照する場合には，メンバー名をデータ・セット名の修飾語としてかつこの中で指示する．たとえば，SYS1. LINKLIB 内の IEHLIST と呼ばれるプログラムを参照したいのであれば，つぎの指定になる：

DSNAME＝SYS1. LINKLIB (IEHLIST)

なお，第6章「データ・セットの編成法とアクセス方式」の第六節「索引順次データ・セットの構成とその処理」の中で説明しておいたように，索引順次データ・セット名を指定するときには，そのデータ・セットの記憶域名 (INDEX, PRIME および OVFLOW) を指定する．索引順次データ・セットで使用する DASD 上の記憶域には，**索引域** (index area)，**主要データ域** (prime data area) および**あふれ域** (overflow area) がある．データ・セットを作成するときは，これらの記憶域を DD ステートメントで定義する必要がある．記憶域名，つまり，INDEX, PRIME または OVFLOW のいずれかの項目をデータ・セット名の直後にかっこで示す．ただし，データ・セット全体を1個の DD ステートメントで定義したいときは，DSNAME＝データ・セット名または

DSNAME＝{
- データ・セット名
- データ・セット名（メンバー名）
- データ・セット名（世代番号）
- データ・セット名 (INDEX または PRIME または OVFLOW)
- && データ・セット名
- && データ・セット名（メンバー名）
- && データ・セット名 (INDEX または PRIME または OVFLOW)
- *．データ定義名
- *．ステップ名．データ定義名
- *．ステップ名．プロシージャー・ステップ名．データ定義名

図 7-66　DSNAME パラメータの一般形式

DSNAME＝データ・セット名 (PRIME) と書けばよい．また，データ・セットを取り出す場合には，INDEX, PRIME および OVFLOW という用語は指定しなくてもよい．

以上説明してきた DSNAME パラメータの一般形式をまとめたのが図 7-66 である．このうち，最後に示した形式についてはここではふれていない．

最後に DSNAME パラメータの具体的な使用例をもとにまとめておこう．図 7-67をみていただきたい．最初の例は，ALPHA という名前の新規データ・セットを定義している．2番目の例では，PDSと名付けられた区分データ・セッ

```
1.  // DD 1      DD     DSNAME=ALPHA, DISP=(, KEEP),
                                          UNIT=2400,      ×
    //                  VOLUME=SER=044888
2.  // DD 2      DD     DSNAME=PDS (PROG 12), DISP=(OLD,
                                     KEEP), UNIT=2311     ×
    //                  VOLUME=SER=882334
3.  // DD 3      DD     DSNAME=&&WORK, UNIT=2400
4.  // STEP 1    EXEC   PGM=CREATE
    // DD 4      DD     DSNAME=&&ISDATE (PRIME), DISP
                              =(, PASS), UNIT=(2311, 2)   ×
    //                  SPACE=(CYL, (10, , 2), , CONTIG),
                              VOLUME=SER=(33489, 33490)
    // STEP 2    DD     PGM=OPER
    // DD 5      DD     DSNAME=*. STEP 1. DD 4, DISP
                                       =(OLD, DELETE)
```

図 7-67　DSNAME パラメータの使用例

トからメンバーを取り出している．次の例は，一時的データ・セットを定義している．データ・セットはジョブの終りには削除される性質のものとなっているので，DSNAME を省略しても一向にさしつかえはない．最後の例の STEP 1 の DD 4 と名付けられた DD ステートメントでは，ISDATA という名前がつけられた一時的索引順次データ・セットを定義している．このDD ステートメントは，索引順次データ・セットの全域を定義するのに使われている．STEP 2 の DD 5 と名付けられた DD ステートメントでは，先行ステップで定義された DD ステートメントを参照して，このデータ・セットを使用しようとしている．STEP 1 でデータ・セットはパスされる性質のものであるとしているので，STEP 2 でこのデータ・セットが取り出せるのである．

3) 装置のタイプに関する指定

そのジョブで使用するデータ・セットがどのようなタイプの装置に収容されているか，ないしはどのようなタイプの装置に収容しようとしているかをシステムに知らせる必要がある．この情報にもとづいて開始プログラムが入出力装置の割り当ての機能を果してくれるのである．この種の情報を指示するのに用いられるのが UNIT パラメータである．UNIT パラメータでの代表的指定内容はつぎの通りである：

$$\text{UNIT}=\begin{Bmatrix}\text{装置タイプ}\\ \text{装置アドレス}\\ \text{グループ名}\end{Bmatrix}$$

記号 { } は，この中のいずれか1つを指定することを意味している．つまり，UNIT パラメータでは，装置タイプ，装置アドレス，グループ名のいずれかを指定する．

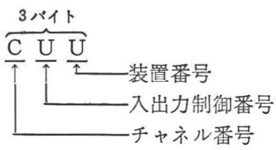
① 装置タイプの指定

UNIT パラメータで装置のタイプを指定するときは，たとえば3420磁気テープ装置であれば UNIT=3400-3, 3330 磁気ディスク装置であれば UNIT=3330, 2314 磁気ディスク集団装置であれば UNIT=2314 といった具合いに指定する．

つまり，装置ごとに定められている個有の番号で指定する方法である．このような装置タイプで指示しておくと，開始／終止プログラムは指定された装置タイプの中から空いている装置を見つけ出して該当データ・セットに割り当ててくれる．

② 装置アドレスの指定

装置アドレスは，3バイトからなり，下記に示すようにチャネル番号，入出力制御装置番号および装置番号とから成り立っている：



この装置アドレスは，ハードウェア・システムが導入されるときに物理的に決まってしまう．したがって，それぞれの導入システムで定められている装置アドレスを用いる必要がある．**図7-68**の中に示している3桁のアドレスが，ある導入システムの装置アドレスの例である．

装置アドレスは，図7-68から想定されるように個々の装置固有の番号である．したがって，UNIT パラメータで装置アドレスを指定すると，その装置しか割り当てが効かない．UNIT＝150 と指定すると 150 の装置が空いていないと当該データ・セットの入出力装置の割り当てができず，その結果このジョブの実行が開始できないことになる．ところが，この図に示されているように装置アドレス 150, 151, 152, 153 の装置は，いずれも3330 磁気ディスク装置である．したがって UNIT＝3330 と装置タイプで指定すれば，いずれかの 3330 磁気ディスク装置が使用可能だと，ジョブの実行が開始できる．つまり，装置アドレスより装置タイプを適用した方が，そのジョブ・ステップが入出力割り当てのために待たされることが少なくなるのである．しかし，代替の装置がなく特定の装置しか要求できないような場合もある．たとえば，図7-68において カード・デックの形式で直接出力しなければならないなら，装置アドレス 00D を持ったカード穿孔装置が一台しかないので，UNIT＝00D と指定せざるを得

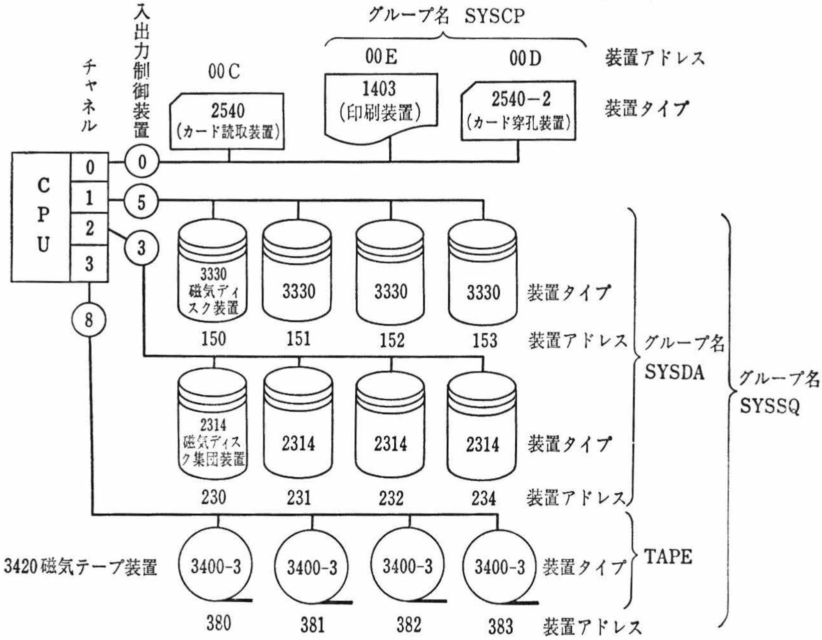


図 7-68　装置アドレス，装置タイプおよびグループ名の相互関係の例

ない．

③ グループ名

グループ名は，オペレーティング・システムを生成するさいに自由につけることが許されている．何らかの共通基準（たとえば，順次処理が可能な装置とか，直接アクセスが効く装置だとか）で，装置をグループ化しそのグループを代表する名前をつける．これが**グループ名**である．たとえば，図7-68では，3330磁気ディスク装置4台と2314磁気ディスク集団装置はいずれも直接アクセスできるという共通的性格があるのでこれらをひとまとめにして，SYSDAというグループ名を与えている．これらの装置に磁気テープ装置を加えると，いずれも順次編成のデータ・セットの入出力ができるという共通的性格があることから，SYSSQとしている．

グループ名の適用は，装置タイプを使用するよりも開始プログラムが入出力装置の選択割当てを行うさいの自由度をより高めることができる．

④ 複数ボリュームに対する装置の割当て要求の指定

データ・セットが複数のボリュームにまたがるような場合には，2台以上の装置の割り当てを要求することもできる．ボリュームのかけかえ時間を最少限にするためには，必要なだけの装置台数をはじめから要求しておいた方がよいことはいうまでもない．このためには，UNIT パラメータの2番目のサブパラメータとして装置の数を指定する．たとえば，3台の磁気テープ装置が必要なら，つぎのように指定すればことたりる：

UNIT＝(3400-3, 3) または UNIT＝(TAPE, 3)

⑤ ボリュームの取付け延期の指定

開始プログラムに**ボリュームの取付け** (volume mount) をさせる必要がないときは，取付け延期を要求することができる．たとえば，特別なテープ・ボリュームが，ジョブ・ステップを開始したあとしばらくの間必要としないのなら，つぎのように指定して，その取付けを延期させることができる：

UNIT＝(TAPE, , DEFER)

取付け延期の要求は，3番目のサブパラメータとして与える．したがって，装置の数を省略した場合にはその旨をカンマで表示する必要がある．なお装置の数が省略されていると，当然のことながら1とみなされる．

⑥ 異ったデータ・セットに別々の装置の割当てを要求する指定

処理効率を向上させるという観点からすると，入出力装置の割当てを最適化するということが重要な課題である．使用頻度の高いデータ・セットを別のボリュームに記録するようにすれば，かなりの効率向上が約束される．たとえば，アセンブラー言語翻訳プログラムを使用するさいには，3種類の作業用データ・セットが必要である（これらのデータ定義名は，SYSUT 1, SYSUT 2, SYSUT 3 と定められている）．この場合，SYSUT 3 のデータ・セットを SYS

UT 1 および SYSUT 2 のデータ・セットとは異ったボリュームに割当てることができると，アクセス・アームの移動に要する待ち時間を大巾に減らすことができる．

こういったとき，直接アクセス・データ・セットに対して別々のアクセス機構の割当てを要求するのに使われるのが，UNIT キーワードの最後のサブパラメータであり，SEP＝で指定する．この SEP ザブパラメータでは，別の DASD ボリュームに割当てて欲しいデータ・セットのデータ定義名を指定する．前述したケースの場合，つまり SYSUT 3 のデータ・セットを SYSUT 1 と SYSUT 2 とは別の装置に割り振って欲しい場合だと，次のように指定するとよい：

```
//SYSUT 1   DD   UNIT=SYSDA
//SYSUT 2   DD   UNIT=SYSDA
//SYSUT 3   DD   UNIT=(SYSDA, SEP=(SYSUT 1, SYSUT 2))
```

⑦ 異ったデータ・セットに同じ装置の割当てを要求する指定

別の要求もでてくる．異ったデータ・セットに別々の装置を割当てるのとは逆に，同じ装置を要求することも考えられる．これは，先述のケースと矛盾すると思われるが，そうではない．相互に関連性のあるデータ・セットは同一のボリュームに収容しておいた方が管理上都合がよい．とくに，コンピュータ室とコンピュータ室との間でデータ・セットの転送をする必要のあるようなときは，このことが要求されよう．ともあれ，この種の要請を満すために UNIT＝AFF＝という形式のキーワード・パラメータが用意されている．つぎの例をみて欲しい：

```
//OBJECT   DD   DSNAME=OBJMODS, DISP=(NEW, KEEP), UNIT=
                3330, ……
//LOAD   DD   DSNAME=LOADMODS, DISP=(NEW, KEEP), UNIT=
              AFF=OBJECT……
```

この場合，データ・セット OBJMODS と LOADMODS の両方が同じボリュ

ームに収容されることになる．UNIT＝AFF＝のサブパラメータとしては，同じ装置を割り当てて欲しいデータ・セットが定義されていてDDステートメントのデータ定義名を用いる．

⑧ 入力データにシステム入力装置の割当てを要求する指定

入力ジョブの流れの中に入れている入力データに**システム入力装置**（system input device）の割当てを要求する方法には2種類ある．いずれの方法でも，UNITパラメータは使用しない．

その1つは図7-69に示しているように＊を使う方法である．DDステートメントのオペランドに＊があると，開始プログラムはこのカードのあとに後くカードを，EXECステートメントで指示されたプログラムで処理するデータ・カードであるとみなす．この星印の効力は，入力ジョブの流れの中の／＊ステートメントないしは別のジョブ制御カードにぶつかったときに失なわれてしまう．つまり，区切りステートメントまたはその他のジョブ制御ステートメントがデータ・セットの終りを示すのである．したがって，この方法を用いる場合には，ジョブ制御ステートメントを入力データの1部とすることはできない．

もう1つの方法は，＊の代りにDATAという単語をオペランドに記入する

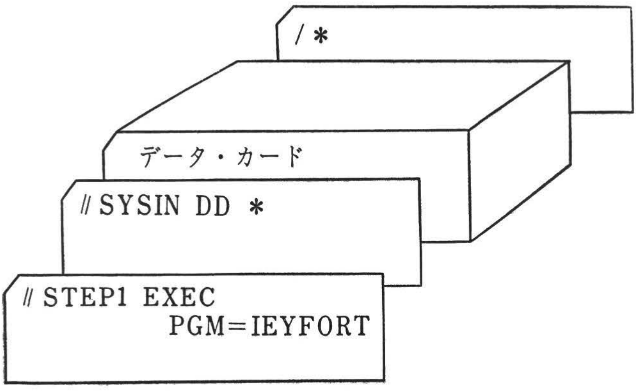


図 7-69 UNITパラメータを使わない特別な例

```
┌UNIT＝(┌装置アドレス┐〔, 装置の数┐〔, DEFER〕〔, SEP＝(データ定義名, …)〕┐
│       │装置タイプ  │〔,        ┘                                        │
│       └グループ名  ┘                                                   │
└UNIT＝AFF＝データ定義名                                                  ┘
```

図 7-70　UNIT パラメータの一般形式

方法である．この場合には，入力データの中にジョブ制御ステートメントが入っていてもその効力は失われない．ジョブ制御ステートメントもデータの1部として扱えるのである．したがって，この方法は，プロシージャーを SYS1. PROCLIB に登録するためにジョブ制御ステートメントを入力するような時に活用される．

以上が装置のタイプに関する指定である．特に UNIT パラメータで指定する場合の一般形式を図 7-70にまとめておいたので参考にして欲しい．

最後に UNIT パラメータの具体的な使用例を示しておこう．図 7-71 がそれ

```
1.  // DD1      DD     DSNAME=AAG3, DISP=(, KEEP),                    ×
    //                 VOLUME=SER=12340, UNIT=2400
2.  // DD2      DD     DSNAME=X.Y.Z, DISP=OLD, UNIT=(, 2)
3.  // DD3      DD     DSNAME=COLLECT, DISP=OLD,                      ×
    //                 VOLUME=SER=1095, UNIT
                                          =(DISK,, DEFER)
4.  // STEP1    EXEC   PGM=XTRA
    // DDA      DD     UNIT=2311, SPACE=(1024, (150, 20))
    // DDB      DD     UNIT=2311, SPACE=(1024, (100, 10))
    // DDC      DD     UNIT=(2311, SEP=(DDA, DDB)),
                                   SPACE=(2048, (300, 30))
5.  // STEP2    EXEC   PGM=POINT
    // DDX      DD     DSNAME=EST, DISP=MOD, VOLUME
                                   =SER=(42569, 42570),           ×
    //                 UNIT=(2311, 2)
    // DDY      DD     DSNAME=ERAS, DISP=OLD,
                                            UNIT=3400-3
    // DDZ      DD     DSNAME=RECK, DISP=OLD,                         ×
    //                 VOLUME=SER=(40653, 13262), UNIT
                                            =AFF=DDX
```

図 7-71　UNIT パラメータの使用例

である．

最初の例では，DD ステートメントで新規データ・セットを定義し，そのデータ・セットに2400磁気テープ装置のうちのどれかを割り振るように要求している．次の例は，カタログされているデータ・セットを定義し，そのデータ・セットには2台の装置を割当てるように開始プログラムに要求している．装置のタイプは省略されているので，カタログからその情報が得られる．**3.** の例は，データ・セットの取付け延期の要求をしている．DD ステートメントで直接アクセス・ボリューム上に収容されているデータ・セットを定義し，DISK というグループ名で代表される装置のどれかを割り当てるよう要求している．DEFER が指示されているので，このボリュームはデータ・セットが OPEN されるまでマウントする必要はない．**4.** の場合には，DD ステートメントで3つの一時的データ・セットを定義している．データ定義名が DDC である DD ステートメントでは，DDA および DDB の DD ステートメントで定義されたデータ・セットに割り当てられる装置以外の2311磁気ディスク装置を割り当ててくれるように要求している．最後の例は，データ定義名 DDZ がつけられた DD ステートメントで，DDX がつけられた DD ステートメントで定義されたデータ・セットに割り当てた装置と同じ装置をこのデータ・セットにも割り当てるように要求している．DDX では，2台の装置を要求しているので，この2台の装置が DDZ で定義されたデータ・セットにも割り当てられることになる．

4）ボリューム通し番号とそれに関連する情報の指定

ボリュームの通し番号をはじめとして，ボリュームにかかわりを持つ種々の属性は VOLUME パラメータで指定する．

① 私的ボリュームの要求を指定

特別なプロジェクトでは，データを記録したり読み取ったりするボリュームをジョブ・ステップが終了した直後に取はずして保管しておきたいといったケースがしばしば生じる．別の処理業務の新規データ・セットのために，このボリュームからスペースを割り振ることを避けたいからである．これを保障し

て，ボリュームを私的所有としてしまうためには，つぎのような指定をする：

VOLUME＝PRIVATE

このような性格がもたされたデータ・セットを特に**私的ボリューム**（private volume）と呼んでいる．ところで，2つ以上のジョブ・ステップにまたがって，私的ボリュームを使用したいときがある．さきの指定だと，ジョブ・ステップが終了すると該当ボリュームはディスマウントされてしまうので，つぎに使用するステップで再びマウントする必要が生じてしまう．これでは，オペレーションに無駄が生じるし，オペレーターの負担にもなる．そこで，そのボリュームを後続するステップが終るまでそのまま保持しておいて欲しいという要求をする必要がある．このためには，VOLUME キーワードの2番目のサブパラメータでRETAINと指定する．つまり，VOLUME＝(PRIVATE, RETAIN)とすればよい．

② 複数ボリューム・データ・セットの特定ボリュームからの開始要求の指定

データ・セットが複数のボリュームにまたがっている場合，その中の特定ボリュームのデータだけを処理したいといった場合がある．通常の指示をしたのでは，ボリューム順序番号が1のものから割り当てられてしまう．したがって，複数ボリューム・データ・セットの特定ボリュームだけを必要とする場合には，その旨指示する必要がある，このためには，VOLUME キーワードの3番目のサブパラメータとして必要とするボリュームのボリューム順序番号を指定する．たとえば，複数ボリューム・データ・セットの3番目のボリュームを要求する場合だと，つぎのようになる：

VOLUME＝(，，3)

③ 2個以上のボリュームを要求する指定

出力データ・セットが複数ボリュームにまたがると想定される場合に，そのボリュームの最大必要数を開始プログラムに知らせるためには，VOLUME キーワードの4番目の定位置サブパラメータを用いる．たとえば，4個のボリュ

ームを割り当てるためには，つぎのような指定になる：

VOLUME=(, , , 4)

④ 特定ボリュームのための指定

データ・セットがカタログされていないときは，DD ステートメントでボリューム通し番号を指定することによって特定のボリュームを要求しなければならない．ボリューム通し番号を指示するやり方には，直接的指示と間接的指示の2通りの方法がある．

直接的指示の方法では，DD ステートメントの中に必要なボリュームの通し番号を直接指定する．たとえば，通し番号 HWS 111，KAC 008 および CCC 039 の通し番号を持った3個のボリュームを必要とする場合だと，つぎのように指定する：

VOLUME=SER=(HWS 111, KAC 008, CCC 039)

1個のボリュームだけ，たとえば HWC 111 だけを必要とするのならつぎのように指定する：

VOLUME=SER=HWC 111

さて，もう1つの例をみておこう．2個の特定の私的ボリューム（通し番号 APC 010 と JCC 006）を要求する場合である．しかも，このボリュームが最初に用いられたステップのあともそのままマウントの状態にしておきたいときは，つぎのようになる：

VOLUME=(PRIVATE, RETAIN, SER=(APC 010, JCC 006))

この例では，最後の2つのサブパラメータ（ボリューム順序番号とボリューム数）が省略されているが，SER サブパラメータがキーワード形式になっているので省略を示すカンマは必要ない．

そのジョブ・ステップで必要となるデータ・セットのボリューム通し番号が分らないときには，**間接的指定の方法**をとる．これには3つの方法がある．

まず，カタログされているデータ・セットと同じボリュームにデータ・セットを収容したい場合である（しかし，ボリューム通し番号は定かでない）．この場合カタログされているデータ・セットの名前が分りさえすれば，通し番号を参照するためにそのデータ・セット名を用いる．たとえば，カタログされているデータ・セット名が SYS 1. COBLIB で，そのデータ・セットと同じボリュームをこのデータ・セットでも使いたいとしたら次のように書く：

```
VOLUME=REF=SYS 1. COBLIB
```

つぎには，特定ボリュームの要求がないデータ・セットが収容されているボリュームと同じボリュームを要求したいときである．たとえば，ジョブ・ステップ4でステップ1のデータ・セットに割り当てられたと同じボリューム上にデータ・セットを書き出したいといったときである（もちろん，ステップ1では特定ボリュームの要求はしていない）．このようなとき，双方のステップで同じボリュームを適用するようにするためには，**逆方向参照**（refer-back）の手法を用いる．逆方向参照の手法とは，先行ステップの DD ステートメントを参照する方法である．この逆方向参照は，＊によって示される．さらに参照するジョブ・ステップと DD ステートメントの名前を指定する必要がある．たとえば，STEPONE の DD 5 と名付けられた DD ステートメントで定義されたデータ・セットと同じボリュームを要求したいのなら，次のような指定になる：

```
//DD 7   DD      …, VOLUME=REF=*. STEPONE. DD 5
```

同一ジョブ・ステップで双方のデータ・セットが新規に作成され，しかも特定ボリュームの指定がなくて同一ボリューム上に2つのデータ・セットを収容したいような場合もある．この場合にはさきの例と異なり，ステップ名を指示する必要はない．同一ボリューム上に収容すべきデータ・セットが定義されている DD ステートメントのデータ定義名を指定すればよい．つぎにその例を示そう：

```
//OUTDD   DD      …, VOLUME=REF*. OUTDDF
```

```
VOLUME=([PRIVATE][, RETAIN][, ボリューム順序番号][, ボリューム数]
                 [,       ][,                 ]
        [, ]{SER=(ボリューム通し番号, ....)                          })
            {REF=データ・セット名                                    }
            {REF=*.データ定義名                                      }
            {REF=*.ステップ名.データ定義名                            }
            {REF=*.ステップ名.プロシージャー・ステップ名.データ定義名    }
```

図 7-72 VOLUMEパラメータの一般形式

OUTDDF がつけられた DD ステートメントでは特定ボリュームの指定はない. この DD ステートメントで定義されたデータ・セットが収容されると同じボリューム上に OUTDD と名付けた DD ステートメントで定義されたデータ・セットも収容するように割り当てを行って欲しい旨要求しているのである.

これまで説明してきた VOLUME パラメータの一般形式を示したのが, **図7-72** である. これをもとに指定内容を各自でまとめてみていただきたい.

VOLUME パラメータの使用例を4つ示したのが**図 7-73**である. **1.** の DD ステートメントでは, すでに存在するデータ・セットを定義し, そのデータ・セットがボリューム通し番号548863に収容されていることを開始プログラムに知らせている. 私的ボリュームであることが指定されているので, 特定ボリュームの指定がないデータ・セットにこのボリュームを割り当てるようなことはなく, このジョブ・ステップでの使用が終るとこのボリュームはディスマウントされてしまう.

2. の例の DD ステートメントでは, 新規データ・セットを定義し, ボリュームの通し番号を指定していないので, 開始プログラムが適当にボリュームを割り当てるよう要求していることになる. UNIT パラメータでは一台の装置しか要求しておらず, しかもボリューム数は2となっているので, 追加のボリュームは必要になった時点でマウントされる.

つぎの例では, DD ステートメントで既存のデータ・セットを定義し, それが通し番号 96341 と 96342 に収容されている旨指示している, さらに, 必要なら合計4ボリュームまでそのデータ・セットの処理に使われることを示している.

最後の例の DD ステートメントでは, 一時データ・セットを定義している

が，VOLUME パラメータが削除されているので，そのデータ・セットに適当なボリュームを割り振るよう開始プログラムに要求していることになる.

5) DASD 上のスペースを要求する指定

DASD上に新規データ・セットを作成するに当っては，そのデータ・セットに必要となるスペースの割当てを開始プログラムに要求しなければならない．このスペースの要求の仕方に関しては，第5章「データ管理の諸機能とその構造」の第七節「直接アクセス・ボリュームのスペース割当て機能」の個所で解説している．DD ステートメントでの割当て要求の指定の方法に関しては，特に「DASD スペースの割当て要求」の項目を参照されたい.

6) チャネル使用の最適化に関する要求

2台以上の入出力チャネルを有している場合だと，それらチャネルを最適使用する必要がある．そのために，異った装置に収容した使用頻度の高いデータ・セットからのデータ・セットの転送を別々のチャネルから行うように要求することができるようにしている．たとえば，LINKLIB と SVCLIB に対しての入出力要求がしばしば同時に行われ，しかもそれぞれのデータ・セットが別々のボリュームに収容されている場合，データ転送を1つのチャネルを使って行うよりも，別々のチャネルからこれら2つのライブラリーのデータを転送したほうがはるかに効率を高めることができよう．この**チャネル分離** (channel separation) を要求するために SEP キーワード・パラメータを使用する．異った

```
1. // DD 1    DD    DSNAME=STEP. UNIT=3330, DISP=OLD,                  ×
   //               VOLUME=(PRIVATE,,, SER=548863)
2. // DDB     DD    DSNAME=COMM, DISP=(NEW, KEEP),
                                        SPACE=(CYL, (30, 2)),     ×
   //               VOLUME=(PRIVATE.,, 2), UNIT=3330
3. // DD 2    DD    DSNAME=QUET, DISP=(MOD, KEEP),
                                              UNIT=(2400, 2),     ×
   //               VOLUME=(,,, 4, SER=(96341, 96342))
4. // DD 3    DD    DSNAME=&&OUT, DISP=NEW, UNIT=3400-3
```

図 7-73　VOLUME パラメータの使用例

```
// STEP 1  EXEC  PGM=START
// DD 1    DD    DSNAME=X. Y. Z, DISP=OLD
// DD 2    DD    DSNAME=&&WORK, DISP=(, PASS),
                                    UNIT=3330,    ×
//               SPACE=(CYL, (3, 1))
// DD 3    DD    DSNAME=NABS, DISP=OLD, VOLUME=
                              SER=7110, UNIT=3330
// DD 4    DD    DSNAME=PARE, DISP=OLD, VOLUME=
                                    SER=E 59,     ×
//               UNIT=3330, SEP=(DD 2, DD 3)
```

図 7-74　SEP パラメータの使用例

チャネルを使用するデータ・セットを指定するのに，そのデータ・セットを定義した DD カード上の DD 名を SEP パラメータで指示する．たとえば，DSONE, DSTWO および DSTREE の 3 つのデータ・セットのそれぞれが ENGLISH, NO 5，HUDSON という名前の DD カードで定義されており，かつ NEWDS と名付けられたデータ・セットとは別のチャネルを使用したいなら，つぎのように指定すればよい：

```
//CHNLSEP  DD  DSNAME=NEWDS, SEP=(ENGLISH, NO 5, HUDSON)
```

図 7-74 に SEP パラメータの使用例を示している．これは，DD 4 と名付けた DD ステートメントで定義したデータ・セットに，DD 2 と DD 3 で定義したデータ・セットに割り当てられるチャネルとは別のものを割り当てて欲しい旨の要求をしている．なお，SEP パラメータではデータ定義名 DD 1 を指示していないので，DD 1 で定義されているデータ・セットは DD 4 で定義されているデータ・セットと同じチャネルが割り当てられるかも知れないし，そうならないかも知れない．

すでにチャネル分離を指示した DD ステートメントのデータ定義名を用いることによって，チャネル分離の要求をすることもできる．この場合には，AFF キーワード・パラメータを使用する．たとえば，つぎのように指定する：

```
//MORESEP DD DSNAME=NEWFL, AFF=CHNLSEP
```

```
// STEP 1     EXEC   PGM=CONVERT
// INPUT 1    DD     DSNAME=A. B. C, DISP=OLD
// INPUT 2    DD     DSNAME=FILE, DISP=OLD, UNIT=3400-3,          ×
//                   VOLUME=SER=54333
// BUF        DD     UNIT=3400-3, SEP=(INPUT 1, INPUT 2)
// OUTPUT     DD     DSNAME=ALPHA, UNIT=TAPE, DISP
                                  =(, KEEP), AFF=BUF
```

図 7-75　AFF パラメータの使用例

これは，さきの例の DD ステートメント CHNLSEP を AFF パラメータで参照している．つぎに示すのと全く同一のことを指示しているのである：

//MORESEP DD DSNAME=NEWFL, SEP=(ENGLISH, NO5, HUDSON)

つまり，AFF パラメータはチャネル分離に関して参照したデータ定義名の場合と同じ性質（このことを channel affinity という）を要求するのである．

AFF パラメータの具体例が図 7-75 である．これは，BUF と OUTPUT の DD ステートメントで定義したデータ・セットに，INPUT 1 と INPUT 2 の DD ステートメントで定義したデータ・セットに割り振るチャネルは別のチャネルを割り当てて欲しいことを要求しているのである．この場合 BUF と OUTPUT の DD ステートメントで定義されたデータ・セットは同一チャネルに割り当てられるかも知れないし，別々のチャネルに割り当てられるかも知れない．なお，AFF=BUF の代りに SEP=(INPUT 1, INPUT 2) としても同じことになる．

7)　データ・セット・ラベルに関する情報の指定

ボリューム・ラベルの情報を指示するのと同様に，データ・セット・ラベルに関する情報を与えることも必要となる．直接アクセス装置の場合だと，データ・セット・ラベルは，**データ・セット制御ブロック** (data set control block) つまり **DSCB** と呼ばれる．すでに第5章で詳述したように，DSCB は VTOC の中に収容されている．

```
LABEL=(〔データ・セット順序番号〕⎧, SL ⎫⎡, PASSWORD ⎤⎡, IN ⎤〔, 〕
                                 ⎪, SUL⎪⎢, NOPWREAD ⎥⎣, OUT⎦
                                 ⎨, NSL⎬⎣,          ⎦
                                 ⎪, NL ⎪
                                 ⎪, BLP⎪           ⎡EXPDT=yyddd⎤)
                                 ⎩,    ⎭           ⎣RETPD=nnnn ⎦
```

図 7-76　LABEL パラメータの一般形式

一方，磁気テープの場合にはボリューム・ラベルは必須ではない．しかも，ボリューム・ラベルがない場合には，データ・セット・ラベルも不要である．また，データ・セット・ラベルには非標準ラベルも許される．これに関しても，第5章の「四．ボリューム・ラベルとデータ・セット・ラベル」の項で詳述した通りである．

データ・セット・ラベルに関する情報は LABEL パラメータで指定する．この LABEL パラメータの一般形式を示したのが図 7-76である．以下でそれぞれのパラメータについて説明してゆく．

DASD 上のデータ・セットの場合だと DSCB を参照することによって，そのデータ・セットが収容されているシリンダーやトラック番号が分り自動的に該当データ・セットが取り出せる．ところが磁気テープの場合にはテープ上の最初に位置するデータ・セットを除いては自動的に取り出すことはできない．そこで，テープ上の何番目に位置するデータ・セットを使用するのかを制御プログラムに知らせる必要がある．このために用いるのが「データ・セット順序番号」である．たとえば，テープ上の4番目のデータ・セットが必要ならば，LABEL＝4　と指示すればよい．

テープ上のラベル・タイプを指定するのには，つぎの定位置サブパラメータを使用する：

SL—ボリューム・ラベルおよびデータ・セット・ラベルが標準形式である．

NSL—ラベルが非標準形式である．

NL—ラベルを用いていない．

SUL—標準のボリューム・ラベルとデータ・セット・ラベル以外にユーザー用データ・セット・ラベルがある．

BLP—ラベルはあるが無視して欲しい場合.

つぎのように指定する. 対象になるデータ・セットがテープ上の3番目のデータ・セットであり, かつ標準ラベルをもっている場合だと,

LABEL=(3, SL)

となる. 一方, テープ上の最初のデータ・セットであり, かつラベルがない場合だと,

LABEL=(, NL)

とすればよい.

データ・セットが作成されたあと, そのデータ・セットの取り出しを合言葉を使って保護する方法については, 第5章の「パスワードによるデータ・セットの保護」の項で説明した. この保護を要求するためには, データ・セットの作成時にLABELパラメータで指定しておく必要がある. このためには, つぎの指定をする：

$$\text{LABEL}=(,\ \text{SL},\ \begin{Bmatrix}\text{PASSWORD}\\ \text{NOPWREAD}\end{Bmatrix}$$

PASSWORDの指示をしておくと, 正しいパスワードが与えられないかぎりこのデータ・セットの読み出し, 変更, 拡張あるいは削除の一切を行うことができなくなる. 一方, NOPWREADの指示があると, このデータ・セットの取出しだけはパスワードを与えなくてもできるが, 変更, 拡張および削除に関しては正しいパスワードが必要となる.

データ・セットが入力処理だけに限られる場合, あるいは逆に出力処理だけに限定されることを示すためには, 定位置サブパラメータ IN または OUT を使用する.

データ・セットを保護する別の方法として用いられるのが, **保存期間** (retention period) ないしは**有効期限** (expiration date) であり RETPD また EXPDT キーワード・サブパラメータを使用する. たとえば, 作成するデータ・

セットに 156 日の保存期間を持たせたい場合には,

LABEL=RETPD=156

と指定すればよい, また, 作成するデータ・セットを1975年 9 月22日まで有効としたい場合だと, 9 月22日は年初から 265 日目に当るので,

LABEL=EXPDT=75265

とすればよい.

LABEL パラメータの使用側を図7-77に示しておいたので参照されたい.

8) データ・セット属性とアクセス方式の要件に関する指定

第 5 章の「データ制御ブロック (DCB) の機能と構成」で説明したようにアクセス方式ルーチンが必要とする情報は, 次の 3 つのものから得られる:

① 問題処理プログラムの中の DCB
② データ・セット・ラベル

```
1. //DD1      DD    DSNAME=HERBI, DISP=(NEW, KEEP),
                                        UNIT=TAPE,        ×
   //               VOLUME=SER=T2, LABEL=(3, NSL,
                                        RETPD=188)
2. //DD2      DD    DSNAME=A.B.C, DISP=(, CATLG,
                              DELETE), UNIT=3400-3,       ×
   //               LABEL=(, NL)
3. //DD3      DD    DSNAME=SPECS, UNIT=3400-3, VOLUME
                                        =SER=10222,       ×
   //               DISP=OLD, LABEL=4
4. //STEP1    EXEC  PGM=FIVE
   //DDX      DD    DSNAME=CLEAR, DISP=(OLD, PASS),
                                        UNIT=3400-3,      ×
   //               VOLUME=SER=1257, LABEL=(, NSL)
   //STEP2    EXEC  PGM=BOS
   //DDY      DD    DSNAME=*.STEP1.DDX, DISP=OLD,
                                        LABEL=(, NSL)
```

図 7-77 LABEL パラメータの使用例

③ DD ステートメントの中の DCB パラメータ

必要な情報が処理プログラムの中で規定された DCB の中にあれば，それを使用する．DCB の中になければ，DD ステートメントの DCB パラメータの中を探す．双方にないとすれば，データ・セット・ラベルからその情報を得るという具合いになっている．

ところで，データ・セットの**状況指定** (disposition) が OLD であると，そのデータ・セットの属性は明確に定められており，すでにデータ・セット・ラベルに記録されているであろう．ただし，データ・セットがラベルをもたない磁気テープに収容されているとすると，その属性のすべてが処理プログラムの中の DCB で定義されていないかぎり，データ・セット属性が自動的に利用できるというわけにはゆかない．このような場合には，必要なデータ・セット属性を外部から指示してやる必要が生じる．つまり，DD ステートメントの DCB パラメータが生きてくるのである．もちろん，このようなときだけ DCB パラメータが必要になるのではない．柔軟性を持たせたプログラムにするためには，その時々に応じたデータ・セット属性を指示できるようにしておいた方が都合のよいことがしばしばある．

DD ステートメントの DCB キーワードで指定する DCB 情報として主に必要となるのは，論理レコード長 (LRECL)，ブロックサイズ (BLKSIZE)，レコード形式 (RECFM)，磁気テープの記録密度 (DEN) などである．また，7トラックの磁気テープを用いる場合には，テープ記録方式 (TRTCH) を指示する必要性が生じることがある．

簡単な例をつぎに示そう：

① 240文字の固定長レコードでブロック化係数が4の物理コードを，データ変換機構つきの7トラックに記録したい場合．ただし，記録密度は1インチ当り 556 文字とする．

```
, DCB=(RECFM=FB, LRECL=240, BLKSIZE=960,
DEN=1, TRTCH=C)
```

② 80文字の固定長，非ブロック化レコードを9トラックのテープに記録したい場合

DCB=(RECFM=F, LRECL=80, BLKSIZE=80)

以上のようにDCBキーワードではデータ・セット属性が指定できるだけではない．アクセス方式ルーチンとかかわりをもつ情報，たとえば入出力バッファの数なども指定できる．言うまでもなく入出力バッファを指定するためには，BUFNOというキーワードを用いる．

なお，DDステートメントでDCB情報を与えなければならないとき，必要な情報と同じDCB情報を持った別なデータ・セットが存在するなら，それを参照させる方法もとれる．ただし，この方法を用いる場合には，参照するデータ・セットの名前がカタログに登録されており，かつ，そのデータ・セットが収容されているボリュームがマウントされていることが前提となる．ともあれ，つぎに示すような指定をすればこと足りる：

DCB=SYS1．LINKLIB

この場合には，LINKLIBデータ・セットの属性をそのまま目下のデータ・セットにも適用したい旨指示しているのである．

時には同じDCB情報を，同一ジョブ内で作成される複数個のデータ・セットに使用しなければならないようなことも生じる．このような場合には，DCB情報は1回だけ定義すればよい．そのあとは，DCB情報が最初に指示されたDDステートメントを参照する方法が認められている．たとえば，STEPTWOと名付けられたジョブ・ステップのORIGDCBというデータ定義名のDDカードでDCBパラメータが与えられており，このDCBパラメータを利用したいとするとつぎのように指定する：

DCB=*．STEPTWO．ORIGDCB（別のステップで参照するとき）

または

DCB=*．ORIGDCB（同一ジョブ・ステップで参照するとき）

//名前欄	命令欄	オ ペ ラ ン ド	P/K	備 考
// 〔データ定義名〕	DD	[＊ | DATA]	P	入力ジョブの流れの中にあるデータ・セットを定義.
		〔DUMMY〕	P	特定データ・セットの入出力オペレーションをバイパスするために指定.
		DISP＝({NEW | OLD | MOD | SHR}{, DELETE | , PASS | , KEEP | , CATLG | , UNCATLG}{, DELETE | , KEEP | , CATLG | , UNCATLG})	K	データ・セットの状況，後始末および条件付後始末の指定のために使う.
		DSNAME＝{データ・セット名 データ・セット名（メンバー名） データ・セット名（世代番号） データ・セット名 (INDEX または PRIME または OVERFLOW) && データ・セット名 && データ・セット名（メンバー名） && データ・セット名 (INDEX または PRIME または OVERFLOW) ＊.データ定義名 ＊.ステップ名.データ定義名 ＊.ステップ名.プロシージャ・ステップ名.データ定義名}	K	新規データ・セットに名前を割り当てたり，既存のデータ・セットを識別するのに用いる.
		{UNIT＝({装置アドレス | 装置タイプ | グループ名}[, 装置の数 | ,]〔, DEFER〕〔, SEP＝(データ定義名, …)〕) UNIT＝AFF＝データ定義名}	K	装置に関する情報をシステムに与える.
		[VOLUME＝(〔PRIVATE＝[, RETAIN | ,][, ボリューム順序番号 | ,]〔, ボリューム数〕〔, 〕 {SER＝(ボリューム通し番号, …) REF＝データ・セット名 REF＝＊.データ定義名 REF＝＊.ステップ名.データ定義名 REF＝＊.ステップ名.プロシージャー・ステップ名.データ定義名}	K	ボリュームに関する情報をシステムに与える.

		¹SPACE=({TRK | CYL | ブロックサイズ}, (数量, [, 増分 | ,] [, ダイレクトリー | , 索引]) [, RLSE | ,] [, CONTIG | , MXIG | , ALX | ,] [, ROUND] ²SPACE=(ABSTR, (数量, アドレス [, デイレクトリー | , 索引]))	K	1. 新規データ・セットに対してDASDボリュームのスペースを要求する. 2. 新規データ・セットに対してDASDボリューム上の特定トラックの割り当てを要求する.
		[SPLIT={N | (N, CYL, (数量[, 増分])) | % | (%, ブロックサイズ, (数量[, 増分]))}]	K	新規データ・セットに対してDASDボリュームのスペースを要求する. データ・セットはシリンダーを共有する.
		[SUBALLOC=({TRK | CYL | ブロックサイズ}, (数量 [, 増分 | ,] [, デイレクトリー]) [, データ定義名 | , ステップ名.データ定義名])]	K	先行ジョブ・ステップで割り当てられたDASDボリュームのスペースの一部を要求する.
		[SEP=(データ定義名, …)]	K	チャネル分離を要求
		[AFF=データ定義名]	K	チャネル分離を要求する別の方法.
		[LABEL=([データ・セット順序番号] {, SL | , SUL | , NSL | , NL | , BLP | ,} [, PASSWORD | , NOPWREAD | ,] [, IN | , OUT] [,] [EXPDT= | RETPD=])]	K	ラベルに関する情報を与える.
		[DCB=(データ・セットの属性を列記) | DCB=({データ・セット名 | *.データ定義名 | *.ステップ名.データ定義名} [, 属性の列記])]	K	データ制御ブロック(DCB)を完成させるに必要な情報を与える.

凡例：P—定位置パラメータ
K—キーワード・パラメータ
{ }—いずれか1つを選択
[ ]—任意指定. 1つだけ選択するか全く選択しない.

表 7-8 DD ステートメントの要約（主要なパラメータだけ）

```
1.  // DD1   DD   DSNAME=ALP, DISP=(, KEEP), VOLUME
                                          =SER=44321,     ×
    //            UNIT=2400, DCB=(RECFM=FB, LRECL=240,
                                          BLKSIZE=960,    ×
    //            DEN=1, TRTCH=C)
2.  // DD2   DD   DSNAME=BAL, DISP=OLD, DCB=(RECFM=F,
                                          LRECL=80,       ×
    //            BLKSIZE=80)
    // DD3   DD   DSNAME=CNANN, DISP=(, CATLG, DELETE),
                                          UNIT=2400,      ×
    //            LABEL=(, NL), VOLUME=SER=663488,
                                          DCB=*. DD2
3.  // DD4   DD   DSNAME=JST, DISP=(NEW, KEEP), UNIT=2311,    ×
    //            SPACE=(CYL, (12, 2)), DCB=(A. B. C, KEYLEN=8)
4.  // DD5   DD   DSNAME=SAME, DISP=OLD, UNIT=2311,           ×
                  DCB=(*. STEP1. DD8, BUFNO=5)
```

図 7-78　DCB パラメータの使用例

これまで説明してきたデータ・セット属性とアクセス方式に関する要件の指定の仕方をまとめたのが，図 7-78の例である．

1. の例は，新規作成データ・セットを定義し，かつデータ制御ブロック（DCB）を完成するに必要な情報を指示している．

2. の例は，前に定義した DD ステートメントの中の DCB サブパラメータを参照する例である．つまり，DD3 のステートメントでは新規作成データ・セットを定義し，かつ，同一ジョブ・ステップ内の DD2 のステートメントから DCB サブパラメータをそのままコピーするように要求しているのである．

3. の例は，A. B. C という名前でカタログされているデータ・セットのデータ・セット・ラベルから DCB 情報をコピーするよう要求している．ただし，A. B. C のデータ・セット・ラベルにキーの長さが示されていても，DD4 のステートメントで KEYLEN を指示しているのでこれが優先的に適用される．

4. の例は，DD ステートメントで既存のデータ・セットを定義し，かつ，同一ジョブの別のステップで定義されている DD8 のステートメントから DCB サブパラメータをコピーするよう要求している．ただし，BUFNO はこの DD ステートメントの DCB パラメータの中で指示しているのでこれが有効となる．

以上が，DD ステートメントで指定することのできる主要なパラメータである．これまで説明してきたパラメータの要約を表 7-8 に掲げておいたので参考にしていただきたい．

JOB ステートメント，EXEC ステートメント，DD ステートメントを中心にジョブ制御ステートメントの機能構造を説明してきた．この説明によってジョブ制御ステートメントが作成できるようになることを目標としたのではない．ジョブ管理の機能を実現してゆくために，外部からどのような情報をどのような形式で指示する必要があるかを具体的に把握してもらうことをその主眼としてきた．これを通じて，ジョブ制御ステートメントの機能構造を明らかにすると同時にジョブ・スケジューラーの果す役割りの裏づけとしたかったのである．したがって，特殊なジョブ制御ステートメントや特殊なパラメータに関してはその具体的説明は省略した．

索　引

あ　行

か　行

さ　行

た　行

な　行

は　行

ま　行

や　行

ら　行

≪著者略歴≫

江村潤朗（えむらじゅんろう）

昭和13年　広島県三和町にて生まれる

昭和37年　早稲田大学政治経済学部卒業

同年4月　日本アイ・ビー・エム株式会社に入社．カストマー教育のインストラクターを約5年間担当．昭和43年よりコンピュータ教育の手法や教材の開発にあたる

現　在：教育センター・担当マネジャー・アシスタント

著　書：「すぐに役立つコンピュータ入門」（日本工業新聞社）

主要論文　「企業内におけるコンピュータ要員教育——その問題点と将来の方向——」（情報処理 Vol. 12, No. 11 (1971) p. 731〜734）

その他教育工学に関するもの，企業内コンピュータ教育の進め方に関するもの，プログラマーの適性に関するものおよびコンピュータのハードウェアとソフトウェアに関するものなどがある

現住所　川崎市高津区向ケ丘 430
宮前平グリーンハイツ35—104

オペレーティング・システムへの構造的アプローチ（中巻）

昭和50年9月15日　初版発行
昭和50年12月10日　2版発行
昭和52年4月20日　3版発行

著　者　江　村　潤　朗
発行者　村　越　勝　弘
発行所　**日本コンピュータ協会**
東京都文京区本郷2丁目15—13
電話　東京 (03) 813—6929
振替口座　東京96313番